

目录

介绍

修订记录	1.1
内容提要	1.2
术语表	1.3

说明

使用说明	2.1
模块说明	2.2
示例说明	2.3
编译说明	2.4
注意事项	2.5

示例运行

示例运行说明	3.1
autoplt-demo	3.2
scheduler-demo	3.3
sensor-abstraction-demo	3.4
fusion-demo	3.5
http-server-demo	3.6
local-visualizer-demo	3.7
recorder-demo	3.8
ros1_bridge-demo	3.9
ros2_bridge-demo	3.10
transform-demo	3.11
transform_util_demo	3.12
camera-fusion-demo	3.13
radar-fusion-demo	3.14
adstime-demo	3.15
task-manage-demo	3.16
communication-safety-demo	3.17
diagnostic-demo	3.18
timed	3.19
sensor-manager-demo	3.20

工具使用

工具使用说明	4.1
Monitor工具	4.2
SysMo工具	4.3
Record工具	4.4
诊断转换工具	4.5

- [BST-Platform SDK用户指南](#)
 - [修订记录](#)

BST-Platform SDK用户指南

修订记录

版本号	修订记录	修订日期	作成者
v0.2.1	初稿作成	2020/09/03	BST-Platform开发组
v0.8.0	修复bug和文档遗漏；新增特性	2020/09/15	BST-Platform开发组
v0.8.2	子章节格式调整	2020/09/24	BST-Platform开发组
v0.8.7	更新E2E以及Fusion相关用户指南	2020/10/14	BST-Platform开发组
v0.8.9	更新诊断功能用户指南；更新profile 04使用指南	2020/10/23	BST-Platform开发组
v0.9.0	更新通信模块和传感器融合的用户指南	2020/11/20	BST-Platform开发组
v0.9.1	新增security证书生成指南	2020/12/17	BST-Platform开发组
v1.0.0	更新文档格式	2021/06/24	BST-Platform开发组

版权信息

本文件涉及之信息，属黑芝麻智能科技（上海）有限公司所有。

未经黑芝麻智能科技（上海）有限公司允许，文件中的任何部分都不能以任何形式向第三方散发。

黑芝麻智能科技（上海）有限公司完全拥有知识产权，并受国际知识产权法律保护。

- [内容提要](#)

内容提要

BST-Platform SDK是黑芝麻智能科技基于**HS**系列嵌入式开发平台推出的一款智能驾驶平台**SDK**开发包，旨在帮助开发者快速开发出智能驾驶应用并完成部署。其主要功能是结合**HS**系列芯片的强大算力，将智能驾驶系统的核心功能模块封装成外围接口开放出来，方便用户快速简便的接入并使用**HS**系列芯片的强大处理能力。

- [术语表](#)

术语表

术语或缩略语	描述
SDK	Software Development Kit, 软件开发工具包
BST-Platform	Black Sesame Technologies Platform缩写，即黑芝麻智能科技平台，由黑芝麻智能科技提供。

- [使用说明](#)

使用说明

BST提供BST-OS安装镜像，SDK，SDK相关开发文档以及配套的各种工具供开发者使用。

BST提供如下几种安装镜像供开发者选用：

镜像	描述
default-image	最小镜像（默认提供）
ros1-bridge-image	最小镜像增加ros1-bridge功能（默认不提供）
ros2-bridge-image	最小镜像增加ros2-bridge功能（默认不提供）
ros1-ros2-bridge-image	最小镜像增加ros1-bridge和ros2-bridge功能（默认不提供）

为保持BST-OS安装镜像尽量轻量级，default-image只集成了BST-Platform的部分必需关键文件，对于其他的一些文件比如示例代码及其执行文件等会全部放到SDK包中，如若需要，开发者可手动将需要的这些文件拷贝到A1000系统中去调试运行。如果开发者需要ROS1-Bridge功能可选用ros1-bridge-image进行部署，如果需要ROS2-Bridge功能可选用ros2-bridge-image进行部署，如果同时需要ROS1-Bridge和ROS2-Bridge的功能可选用ros1-ros2-bridge-image进行部署。

- 模块说明

模块说明

BST提供SDK的docker部署方式，假设SDK版本0.8.1，成功部署docker环境后：

库文件目录：/opt/bstos/0.8.1/sysroots/aarch64-bst-linux/usr/lib64/，

执行文件目录：/opt/bstos/0.8.1/sysroots/aarch64-bst-linux/usr/bin/，

头文件目录：/opt/bstos/0.8.1/sysroots/aarch64-bst-linux/usr/include/，

示例代码目录：/opt/bstos/0.8.1/sysroots/aarch64-bst-linux/usr/include/src/，

以default-image为例说明BST-Platform各个模块，包括模块运行依赖的所有库文件，执行文件，配置文件等，模块是否默认集成进BST-OS。

功能模块	依赖的文件	默认是否集成进BST-OS
通信模块	libcyber.so, libautoplt.so, libautoplt_proto.so, libcyber_proto.so, libautoplt_example_proto.so	是
任务管理	libcyber.so, libexecmgr.so, libtask_proto.so, mainboard	是
传感器融合	libcyber.so, libautoplt.so, libautoplt_proto.so, libtransform_buffer.so, libtransform_wrapper.so, libsensord_fusion_base.so, libsensord_fusion_common.so, libsensord_fusion_proto.so, libcamera_fusion_common.so, libcamera_fusion_proto.so, libcamera_omt_tracker.so, libcamera_multicue_tranformer.so, libsensordfusion_camerafusion_proto.so, libsensord_selection_manager.so, libconfig_manager.so, libradar_fusion_interface.so, libradar_fusion_proto.so, libconti_ars_detector.so, libconti_ars_tracker.so, libconti_ars_preprocessor.so, libsensordfusion_radarfusion_proto.so	是
传感器抽象	libsensord_abstraction_proto.so	是
时间接口	libautoplt.so	是
坐标转换	libtransform_proto.so, libstatic_transform.so, libtransform_broadcaster.so, libtransform_wrapper.so, libtransform_buffer.so, libtf2.so	是
DNPP深度学习后处理模块	无	是
ROS1-Bridge	libcyber.so, libcyber_proto.so, libros_melodic.so	否
ROS2-Bridge	libcyber.so pthread fastrtps cyber_proto librcclcpp.so libstdmsgs.so librcclinterfaces.so librmwimplemente.so librmw.so libmypoco.so librcutils.so librccl.so librosidlgeneratorc.so libbuiltinterfaces.so libyamlparse.so librosgraphmsgs.so librosidltypesupportc.so librosidltypesupportcpp.so librclogging.so	否
示例代码	参考运行章节	否
标定工具	opencv, PCL, NLopt	否

Monitor 工具	libcyber.so, cyber_monitor	是
Recorder 工具	libcyber.so, libcyber_proto.so, cyber_recorder	是
调度 Monitor 工具	libcyber.so, mainboard	是

- 示例说明

示例说明

示例名称	描述
autopl-t-demo	通信模块的示例代码
scheduler-demo	任务调度模块的示例代码
sensor-abstraction-demo	传感器抽象模块的示例代码
fusion-demo	传感器融合模块的示例代码
http-server-demo	http-server示例代码
local-visualizer-demo	本地可视化功能的示例代码
recorder-demo	录制工具的示例代码
ros1_bridge-demo	cyber-ros1桥功能的示例代码
ros2_bridge-demo	cyber-ros2桥功能的示例代码
dnpp-demo	dnpp模块示例代码
transform-demo	坐标转换示例代码
transform-util-demo	坐标转换工具示例代码
camera-fusion-demo	camera fusion示例代码
radar-fusion-demo	radar fusion示例代码
adstime-demo	时钟同步示例代码
task-manage-demo	任务执行管理示例代码
communication-safety-demo	通信安全示例代码
diagnostic-demo	诊断示例代码
timed	timed示例代码
sensor-manager-demo	传感器管理示例代码

- [编译说明](#)
 - [编译示例](#)

编译说明

BST-Platform SDK提供丰富的代码示例供开发者参考，开发者开发时可直接在示例代码的基础上进行修改。假设部署的SDK版本为0.8.1，则示例代码位于/opt/bstos/0.8.1/sysroots/aarch64-bst-linux/usr/include/src/。

前面我们提到，示例代码源文件及其对应的可执行文件，库文件，数据文件，配置文件等被打包进SDK但是并没有集成进BST-OS。

所以如果想在A1000上面运行BST-Platform的编程示例，有如下两种方式：

- 直接从对应版本的SDK中拷贝构建好的示例文件到A1000运行
- 根据示例的源码自行构建示例，然后将构建结果拷贝到A1000运行

编译示例

步骤：

1. 下载并部署A1000 SDK到本机（假设部署SDK版本0.8.1）
2. `cd /opt/bstos/0.8.1/sysroots/aarch64-bst-linux/usr/include/src/`
3. `cd xxxx-demo && mkdir build && cd build`
4. `cmake .. && make`

注意：

1. 构建cyber_ros1_bridge-demo时候需要使用如下cmake命令：`cmake .. -DROS_INC_PATH=${OECORE_TARGET_SYSROOT}/usr/include/`
2. 如果普通账户编译失败，可尝试切换到root账户进行编译。

- 注意事项

注意事项

1. 如果开发的应用用到了 **autopl**（**cyberRT**）的通信功能，编译优化选项用 **O0**。
2. 构建**cyber_ros1_bridge-demo**或者开发的应用用到了**ROS1**相关的头文件，用下面的**CMake**命令：

```
cmake .. -DROS_INC_PATH= ${OECORE_TARGET_SYSROOT}/usr/include/
```
3. 如果前一次构建失败，请将**build**目录下面的文件都删除掉: **rm -rf ***，然后再进行构建

- 示例运行说明

示例运行说明

运行示例代码可通过如下两种方式（假设部署的SDK版本是0.8.1）：

- BST-Platform SDK提供预先构建好的示例可执行文件，位于/opt/bstos/0.8.1/sysroots/aarch64-bst-linux/usr/bin/examples，可直接将待运行示例所需的bin文件，lib文件，配置文件，数据文件，模型文件等拷贝到A1000相应目录下，然后根据使用说明运行示例。
- 用示例代码自行构建，然后将构建后的文件拷贝到A1000上面，根据使用说明运行示例。

将文件拷贝到A1000可通过adb命令或者scp进行拷贝。下面分别介绍如何在a1000上面运行各个示例程序。

- [autopl-t-demo](#)
 - 进程内通信测试
 - 进程间通信测试
 - [TZC](#)通信功能测试
 - 通信优先级功能测试
 - 进程内[Service/Client](#)通信模式功能测试
 - 主机间通信功能测试
 - [DDS security](#) 功能测试
 - [iceoryx](#) 进程间通信功能测试
 - [udp_bridge](#)通信功能测试
 - [autopl-t](#)诊断功能集成测试

autopl-t-demo

运行autopl-t-demo前，需要将全部示例拷贝至A1000对应目录，假设A1000的IP为192.168.2.100，版本号为XXXX，命令如下：

```
scp -r /opt/bstos/XXXX/sysroots/aarch64-bst-linux/usr/bin/examples/autopl-t-demo \
root@192.168.2.100:/usr/bin/examples/
```

再将需要用到的so从SDK中拷贝至A1000上对应位置，命令如下：

```
scp /opt/bstos/XXXX/sysroots/aarch64-bst-linux/usr/lib64/libautopl-t_example_proto.so \
root@192.168.2.100:/usr/lib64
```

然后切换到该目录：

```
cd /usr/bin/examples/autopl-t-demo
```

为程序添加可执行权限：

```
chmod +X *
```

进程内通信测试

- 启动方法

启动一个终端：

在终端输入 `./example_intra_comm`

- 停止方法

`ctrl + C`

- 结果判断

若成功，在控制台会有持续输出，否则，表明测试失败。

进程间通信测试

- 启动方法

启动一个终端，在终端输入 `./example_listener`

启动另一个终端，在终端输入 `./example_talker`

- 停止方法

`ctrl + C`

- 结果判断

若成功，发送端和接收端都有持续输出，否则，表明测试失败。

TZC通信功能测试

- 启动方法

启动一个终端，在终端输入 `./example_tzc_listener`

启动另一个终端，在终端输入 `./example_tzc_talker`

- 停止方法

`ctrl + C`

- 结果判断

若成功，在发送端和接收端都有持续输出，否则，表明测试失败。

通信优先级功能测试

- 启动方法

启动一个终端，

(1) 若不使用优先级配置，输入如下命令：

```
./example_priority n
```

(2) 若使用优先级配置，输入如下命令：

```
./example_priority p
```

- 停止方法

`ctrl + C`

- 结果判断

(1) 不使用优先级配置时，Reader 1，Reader 2，Reader 3，在大部分情况下，依次接收到同一份数据；

(2) 在使用优先级配置时，Reader 1，Reader 2，Reader 3，在大部分情况下，按倒序依次接收到同一份数据；

进程内Service/Client通信模式功能测试

- 启动方法

启动一个终端，在终端输入 `./example_service`

- 停止方法

`ctrl + C`

- 结果判断

如在终端中显示service端和client端都能接收并发送数据，则表明该功能正常可用。

主机间通信功能测试

- 启动方法

确定A主机的IP，对应修改/usr/bin/cyber/setup.bash下CYBER_IP，然后执行如下命令：

```
source /usr/bin/cyber/setup.bash
```

启动数据发送方：

```
./example_talker
```

确定B主机的IP，对应修改/usr/bin/cyber/setup.bash下CYBER_IP，然后执行如下命令：

```
source /usr/bin/cyber/setup.bash
```

启动数据接收方：

```
./example_listener
```

- 停止方法

`ctrl + C`

DDS security 功能测试

运行前，需要将示例拷贝至两块A1000对应目录，假设A1000的IP为IP,版本号为XXXX，命令如下：

```
scp -r /opt/bstos/XXXX/sysroots/aarch64-bst-linux/usr/bin/DDS_Security \
root@IP:/usr/bin/
```

然后切换到该目录：

```
cd /usr/bin/DDS_Security
```

- 配置文件说明

DDS security支持三种场景。第一种场景是支持身份验证和加密，对应的配置文件为SecCfg_AuthEncrypt.xml；第二种场景是支持身份验证，对应的配置文件为SecCfg_AuthOnly.xml；第三种场景是禁身份验证和加密，对应配置文件为SecCfg_NoAuthEncryp.xml。默认配置文件是SecCfg.xml。

- 启用身份验证和加密，需要把SecCfg_AuthEncrypt.xml文件名改为SecCfg.xml，命令如下：

```
cp SecCfg_AuthEncrypt.xml SecCfg.xml
```

- 启用身份验证，需要把SecCfg_AuthOnly.xml文件名改为SecCfg.xml，命令如下：

```
cp SecCfg_AuthOnly.xml SecCfg.xml
```

- 禁用身份验证和加密，需要把SecCfg_AuthEncrypt.xml文件名改为SecCfg.xml，命令如下：

```
cp SecCfg_NoAuthEncrypt.xml SecCfg.xml
```

注意：两个A1000配置必须一致才能通信。

- 启动方法

首先在第一个A1000启动一个终端，终端启动接收方，输入命令

```
./SecureHelloWorldExample subscriber
```

在第二个A1000启动一个终端，终端启动发送方，输入命令

```
./SecureHelloWorldExample publisher 100 10
```

（其中：100是字节大小，最大不超过64619字节；10是发送次数）

- 第一种和第二种场景配置是否成功的判断方法如下：

如xml文件配置错误，则会报错[XMLPARSER Error];

若删除证书（certs文件夹），则会报错[SECURITY_AUTHENTICATION Error];

若修改证书里maincacert.pem，则会报错[SECURITY_AUTHENTICATION Error];

如修改证书里mainpubcert.pem或者mainsubcert.pem，则会报错[SECURITY Error];

如修改证书里mainpubkey.pem或者mainsubkey.pem，则会报错segmentation fault;

- 停止方法

ctrl+C

- 结果判断

不管是否进行security配置，下面的输出结果适用。

第一个A1000发送端输出如下：

```
Starting
message size is:100
Publisher matched
Message index is: 1 SENT
Message index is: 2 SENT
Message index is: 3 SENT
Message index is: 4 SENT
Message index is: 5 SENT
Message index is: 6 SENT
Message index is: 7 SENT
Message index is: 8 SENT
Message index is: 9 SENT
```



```
Message index is: 10 SENT
```

第二个A1000接收端输出如下:

```
Starting
Subscriber running. Please press enter to stop the Subscriber
Subscriber matched
Message 1 RECEIVED
Message 2 RECEIVED
Message 3 RECEIVED
Message 4 RECEIVED
Message 5 RECEIVED
Message 6 RECEIVED
Message 7 RECEIVED
Subscriber unmatched
```

则表明该功能正常可用。注意，由于DDS可能会造成数据丢失，所以可能接收到全部或者部分信息。

若没有上述输出或者打印信息不一样，则说明功能异常。

生成CA证书方法

运行前，需要进入到sdk的docker文件夹中。假设版本号为XXXX，即进入到/opt/bstos/XXXX/sysroots/aarch64-bst-linux/usr/bin/GenerateX509CA/conf文件夹，需要生成一个certs文件夹，命令为：

```
mkdir certs
```

然后运行maincacert.sh 脚本，命令如下：

```
bash maincacert.sh
```

运行结果如下：

```
Generating an EC private key
writing new private key to 'maincakey.pem'
-----
Generating an EC private key
writing new private key to 'mainpubkey.pem'
-----
Using configuration from maincaconf.cnf
Check that the request matches the signature
Signature ok
Certificate Details:
    Serial Number:
        0f:f1:c6:cf:77:9a:65:25:af:01:9d:7b:bf:53:a6:3a:d6:78:03:34
    Validity
        Not Before: Dec 10 09:08:08 2020 GMT
        Not After : Dec  8 09:08:08 2030 GMT
    Subject:
        countryName           = ES
        stateOrProvinceName    = MA
        organizationName       = eProsima
        organizationalUnitName  = eProsima
        commonName             = Main Publisher
        emailAddress            = mainpub@eprosima.com
    X509v3 extensions:
        X509v3 Basic Constraints:
            CA:FALSE
Certificate is to be certified until Dec  8 09:08:08 2030 GMT (3650 days)

Write out database with 1 new entries
Data Base Updated
```

```

Generating an EC private key
writing new private key to 'mainsubkey.pem'
-----
Using configuration from maincaconf.cnf
Check that the request matches the signature
Signature ok
Certificate Details:
    Serial Number:
        0f:f1:c6:cf:77:9a:65:25:af:01:9d:7b:bf:53:a6:3a:d6:78:03:35
    Validity
        Not Before: Dec 10 09:08:08 2020 GMT
        Not After : Dec  8 09:08:08 2030 GMT
    Subject:
        countryName           = ES
        stateOrProvinceName   = MA
        organizationName      = eProsima
        organizationalUnitName = eProsima
        commonName            = Main Subscriber
        emailAddress          = mainsub@eprosima.com
    X509v3 extensions:
        X509v3 Basic Constraints:
            CA:FALSE
Certificate is to be certified until Dec  8 09:08:08 2030 GMT (3650 days)

Write out database with 1 new entries
Data Base Updated

```

出现上述结果说明生成X.509的CA证书成功。

如果出现以下结果

```

Generating an EC private key
writing new private key to 'maincakey.pem'
-----
Generating an EC private key
writing new private key to 'mainpubkey.pem'
-----
Using configuration from maincaconf.cnf
Check that the request matches the signature
Signature ok
ERROR:There is already a certificate for /C=ES/ST=MA/O=eProsima/OU=eProsima/CN=Main \
Publisher/emailAddress=mainpub@eprosima.com
The matching entry has the following details
Type          :Valid
Expires on    :301208090808Z
Serial Number :0FF1C6CF779A6525AF019D7BBF53A63AD6780334
File name     :unknown
Subject Name   :/C=ES/ST=MA/O=eProsima/OU=eProsima/CN=Main Publisher/emailAddress=mainpub@eprosima.com
Generating an EC private key
writing new private key to 'mainsubkey.pem'
-----
Using configuration from maincaconf.cnf
Check that the request matches the signature
Signature ok
ERROR:There is already a certificate for /C=ES/ST=MA/O=eProsima/OU=eProsima/CN=Main \
Subscriber/emailAddress=mainsub@eprosima.com
The matching entry has the following details
Type          :Valid
Expires on    :301208090808Z
Serial Number :0FF1C6CF779A6525AF019D7BBF53A63AD6780335
File name     :unknown
Subject Name   :/C=ES/ST=MA/O=eProsima/OU=eProsima/CN=Main Subscriber/emailAddress=mainsub@eprosima.com

```

需要删除index.txt里面的内容。

生成CA证书后DDS Security测试

- 把生成maincacert.pem、mainpubkey.pem、mainpubcert.pem、mainsubkey.pem、mainsubcert.pem替换A1000上DDS_Security/certs/文件夹下的文件。
- 修改A1000时间。由于A1000一般没有进行时间同步，所以A1000时间没有在证书有效时间范围内。

首先查看A1000时间，命令为：

```
date
```

再查看证书有效时间范围，查看mainpubcert.pem和mainsubcert.pem文档中的 Validity开始和结束时间

```
vi mainpubcert.pem
vi mainsubcert.pem
```

然后关闭A1000的时间同步功能，命令如下：

```
timedatectl set-ntp no
```

最后需要把A1000的系统时间修改在证书有效时间内，命令如下：

```
timedatectl set-time 'YYYY-MM-DD HH:MM:SS'
```

- 按照上面的启动方法重新测试DDS Security功能

iceoryx 进程间通信功能测试

配置文件使用方法：

在/usr/bin/TestCases/iceoryx_test目录下，存在文件名为《roudi_config_example.toml》的配置文件，用于配置在程序初始化时，预先分配的内存块大小和数量，文件内容如下：

```
# Adapt this config to your needs and rename it to e.g. roudi_config.toml
[general]
version = 1

[[segment]]

[[segment.mempool]]
size = 128
count = 10000

[[segment.mempool]]
size = 1024
count = 5000

[[segment.mempool]]
size = 16384
count = 1000

[[segment.mempool]]
size = 131072
count = 200

[[segment.mempool]]
size = 524288
count = 50

[[segment.mempool]]
size = 1048576
count = 30
```

```
[[segment.mempool]]
size = 4194304
count = 10
```

每一个segment.mempool代表一种大小的内存配置，其中size表示内存大小，以字节为单位；count表示数量。

在/usr/bin/目录下使用如下命令启动iox-roudi时，会加载配置文件：

```
./iox-roudi -c /usr/bin/TestCases/iceoryx_test/roudi_config_example.toml
```

使用配置文件设置初始化内存大小后，可通过控制台输出，验证设置是否成功：

```
Reserving XXXX bytes in the shared memory [/root]
```

通过观察XXXX的数值变化，可验证设置生效与否。

一对一进程间通信测试方法：

- 启动方法

启动一个终端，连接到A1000，切换到usr/bin目录，输入./iox-roudi，启动iox-roudi中间程序。

启动另一个终端，连接到A1000，切换到usr/bin/TestCases/iceoryx_test目录，输入

```
./iox-ex-subscriber-simple X sub_1_radar Radar
```

X（X可取值为【1,2,3】，分别表示小数据、中数据、大数据，所设参数必须与另一端保持一致），启动数据接收端；"sub_1_radar"表示节点名称，需保证每个节点名称唯一；"Radar"表示接收的数据类型，需与数据发送端匹配。

启动另一个终端，连接到A1000，切换到usr/bin/TestCases/iceoryx_test目录，输入

```
./iox-ex-publisher-simple X pub_1_Radar Radar
```

X可取值为【1,2,3】，分别表示小数据、中数据、大数据，所设参数必须与另一端保持一致；"pub_1_Radar"表示节点名称，需保证每个节点名称唯一；"Radar"表示发送的数据类型，需与数据发送端匹配。

- 停止方法

在启动iox-roudi的终端中，使用Ctrl+C关闭所有程序。

- 结果判断

程序正常启动后，会在第二个和第三个启动的终端中有打印输出，表示正在收发数据，同时，会在同目录生成文件《XXXX.txt》，XXXX表示设置的节点名称，文件中包含3列，第一列为当前时间，第二列为消息序号，第三列为通信时间（单位为纳秒）。

多对多进程间通信测试方法：

在可执行程序同级目录下，存在名为"multi_nodes_test.sh"的脚本文件，文件内容如下：

```
./iox-ex-subscriber-simple 1 sub_1_radar Radar &
./iox-ex-subscriber-simple 1 sub_2_radar Radar &
./iox-ex-subscriber-simple 1 sub_3_camera Camera &

./iox-ex-subscriber-simple 3 sub_4_lidar Lidar &
```

```
./iox-ex-subscriber-simple 3 sub_5_lidar Lidar &

./iox-ex-publisher-simple 1 pub_1_Radar Radar &
./iox-ex-publisher-simple 1 pub_2_Camera Camera &
./iox-ex-publisher-simple 3 pub_3_Lidar Lidar
```

用户可参考该脚本文件，配置任意数量的数据收发方，并测试通信性能。

`iox-ex-subscriber-simple`和`iox-ex-publisher-simple`表示要执行的程序，分别代表数据订阅方和数据发送方；第二个参数可选【1,2,3】，表示数据大小，意义同上；第三个参数表示节点名称，必须保证每一个节点名称不同；第四个参数表示数据的通道，接收方和发送方的通道必须相同，才能相互通信。

- 脚本启动方式：

首先打开一个终端，切换到`/usr/bin`目录，输入`./iox-roudi`，启动`iox-roudi`中间程序。

再打开另外一个终端，切换到`/usr/bin/TestCases/iceoryx_test`下，执行脚本：

```
bash multi_nodes_test.sh
```

- 程序停止方式：

使用`Ctrl+C`停止`iox-roudi`

- 结果分析

每一个数据接收端，会生成一个文件，文件名称类似《`sub_1_radar.txt`》，文件中包含3列，第一列为当前时间，第二列为消息序号，第三列为通信时间（单位为纳秒）。

udp_bridge通信功能测试

- 启动方法

启动一个终端，连接到A1000，切换到`/usr/bin/TestCases/`目录，输入`./test_udp_bridge_receiver`。

启动另一个终端，连接到A1000，切换到`/usr/bin/TestCases/`目录，输入`./test_udp_bridge_sender`。

- 停止方法

在两个终端分别输入`Ctrl+C`，可停止对应程序。

- 结果判断

程序正常启动后，会在两个启动的终端中有打印输出，表示正在收发数据。

autoplt诊断功能集成测试

- 运行准备

新建一个终端1，拷贝诊断配置表至A1000对应目录（若诊断配置表已经拷贝过，可忽略本步骤操作）：

```
cd /opt/bstos/XXXX/sysroots/x86_64-bstsd-linux/usr/bin/native-tools/diagnostic_conversion/output
adb push diagnostic_did.bin /etc/diagnostic_bin
adb push diagnostic_dtc.bin /etc/diagnostic_bin
```

注意：每次`push diagnostic_did.bin`和`diagnostic_dtc.bin`完成后，需重启A1000以覆盖原来文件。

确保诊断服务启动,查看诊断服务状态：

```
systemctl status diagnose-service
```

```
[[0;1;32m[[0m diagnose-service.service - DiagnoseService systemd service.
Loaded: loaded (/lib/systemd/system/diagnose-service.service; disabled; vendor preset: enabl
ed)
Active: [[0;1;32mactive (running) [[0m since Thu 2020-11-19 02:32:17 UTC; 7min ago
Main PID: 413 (DiagnoseService)
Tasks: 8 (limit: 2152)
Memory: 5.7M
CGroup: /system.slice/diagnose-service.service
└─413 /usr/bin/DiagnoseService
```

- 启动方法

新建另一个终端2，连接到A1000，切换到/usr/bin/examples/autopltdemo/目录，

```
VSOMEIP_CONFIGURATION=/etc/bstos_service/diagnosticservice/vsomeip-diag.json ./example_diagnostic
```

- 停止方法

在终端执行Ctrl+C，可停止测试程序。

- 结果判断

程序正常启动后，会在终端2有打印输出，类似于以下内容：

```
autopltd report com state: COMSTATE_SHMFAILED
ComErrorCallback recv com_state::ComAbnormalState::COMSTATE_SHMFAILED
autopltd report com state: COMSTATE_SDFAILED
ComErrorCallback recv com_state::ComAbnormalState::COMSTATE_SDFAILED
autopltd report com state: COMSTATE_MSGWRONG
ComErrorCallback recv com_state::ComAbnormalState::COMSTATE_MSGWRONG
autopltd report com state: COMSTATE_SENDFAILED
ComErrorCallback recv com_state::ComAbnormalState::COMSTATE_SENDFAILED
autopltd report com state: COMSTATE_RECVFAILED
ComErrorCallback recv com_state::ComAbnormalState::COMSTATE_RECVFAILED
autopltd report com state: COMSTATE_MSGLOSE
ComErrorCallback recv com_state::ComAbnormalState::COMSTATE_MSGLOSE
autopltd report com state: COMSTATE_OK
```

- **scheduler-demo**
 - 启动一个终端执行如下命令（假设SDK版本号为0.8.1）
 - 启动三个终端分别执行如下命令
 - 运行

scheduler-demo

启动一个终端执行如下命令（假设SDK版本号为0.8.1）

1. adb shell
2. mkdir -p /usr/bin/cyber/dag && exit
3. cd /opt/bstos/0.8.1/sysroots/aarch64-bst-linux/usr/bin/examples
4. adb push scheduler-demo/conf/classic/* /usr/bin/cyber/conf
5. adb push scheduler-demo/dag/* /usr/bin/cyber/dag
6. adb push scheduler-demo/data/* /usr/bin
7. cd /opt/bstos/0.8.1/sysroots/aarch64-bst-linux/usr/lib64
8. adb push libperception_mock.so libfusion_mock.so libplan_mock.so libschedproto.so /usr/lib64

启动三个终端分别执行如下命令

1. adb shell
2. cd /usr/bin && source cyber/setup.bash

运行

1. 终端1执行 mainboard -d perception.dag -p perception-refapp
2. 终端2执行 mainboard -d fusion.dag -p fusion-refapp
3. 终端3执行 mainboard -d plan.dag -p plan-refapp

- [sensor-abstraction-demo](#)

sensor-abstraction-demo

demo可执行文件名为 `sensor_abstraction_demo`

如果开发板系统中 `/usr/bin/examples/sensor-demo` 目录下没有名为 `sensor_abstraction_demo`的可执行文件，通过如下方法将可执行文件从SDK中拷贝到开发板系统中

```
cd {SDK_INSTALL_DIR}/sysroots/aarch64-bst-linux/usr/bin/examples
adb push sensor_abstraction_demo /usr/bin/
```

注意，demo基于cyber库开发，运行demo之前需要执行如下命令

```
source /usr/bin/cyber/setup.bash
```

demo启动命令如下：

```
chmod +x sensor_abstraction_demo
./sensor_abstraction_demo
```


- **fusion-demo**
 - 拷贝示例至A1000
 - 在A1000上运行示例

fusion-demo

fusion-demo 提供多传感器数据融合示例。

fusion-demo位于/opt/bstos/XXXX/sysroots/aarch64-bst-linux/usr/bin/examples/fusion-demo目录下。

拷贝示例至A1000

1.本demo已经集成诊断功能，需拷贝诊断配置表至A1000对应目录（若诊断配置表已经拷贝过，可忽略本步骤操作）：

```
cd /opt/bstos/XXXX/sysroots/x86_64-bst-linux/usr/bin/native-tools/diagnostic_conversion/output
adb push diagnostic_did.bin /etc/diagnostic_bin
adb push diagnostic_dtc.bin /etc/diagnostic_bin
```

注意：每次push diagnostic_did.bin和diagnostic_dtc.bin完成后，需重启A1000以覆盖原来文件。

2.运行fusion-demo前，需要将示例拷贝至A1000对应目录：

```
cd /opt/bstos/XXXX/sysroots/aarch64-bst-linux/usr/bin/examples
adb shell mkdir -p /usr/bin/examples
adb push fusion-demo /usr/bin/examples
cd /opt/bstos/XXXX/sysroots/aarch64-bst-linux/usr/lib64
adb push libfusion_component_demo.so /usr/lib64
```

3.切换到A1000该目录,为程序添加可执行权限：

```
运行adb shell进入a1000的shell环境
cd /usr/bin/examples/fusion-demo && chmod +X *
```

在A1000上运行示例

1.确保诊断服务启动,查看诊断服务状态：

```
运行adb shell进入a1000的shell环境
systemctl status diagnose-service
```

```
[[0;1;32m[[0m diagnose-service.service - DiagnoseService systemd service.
Loaded: loaded (/lib/systemd/system/diagnose-service.service; disabled; vendor preset: enabled)
Active: [[0;1;32mactive (running) [[0m since Thu 2020-11-19 02:32:17 UTC; 7min ago
Main PID: 413 (DiagnoseService)
Tasks: 8 (limit: 2152)
Memory: 5.7M
CGroup: /system.slice/diagnose-service.service
└─413 /usr/bin/DiagnoseService
```

2.新建一个终端1，依次执行以下步骤：

```
运行adb shell进入a1000的shell环境
cd /usr/bin/examples/fusion-demo
source /usr/bin/cyber/setup.bash
```

```
VSOMEIP_CONFIGURATION=/etc/bstos_service/diagnosticservice/vsomeip-diag.json \  
mainboard -d ./example_fusion.dag
```

3.新建一个终端2，依次执行以下步骤：

```
运行adb shell进入a1000的shell环境  
source /usr/bin/cyber/setup.bash  
cd /usr/bin/cyber/tools/cyber_recorder  
./cyber_recorder play -f \  
/usr/bin/examples/fusion-demo/test-data/fusion-demo-test-data.record
```

4.此时终端1开始目标融合处理，终端窗口不断打印出处理结果及耗时情况。

- [http-server-demo](#)
 - 运行http-server-demo
 - 浏览器访问

http-server-demo

http-server-demo演示了通过继承HttpServer类，可以处理HTTP请求，还可以通过WebSocket协议主动向浏览器端发送数据。把HttpServer集成到运行在芯片应用上，就可以通过HTTP协议控制芯片应用的行为。

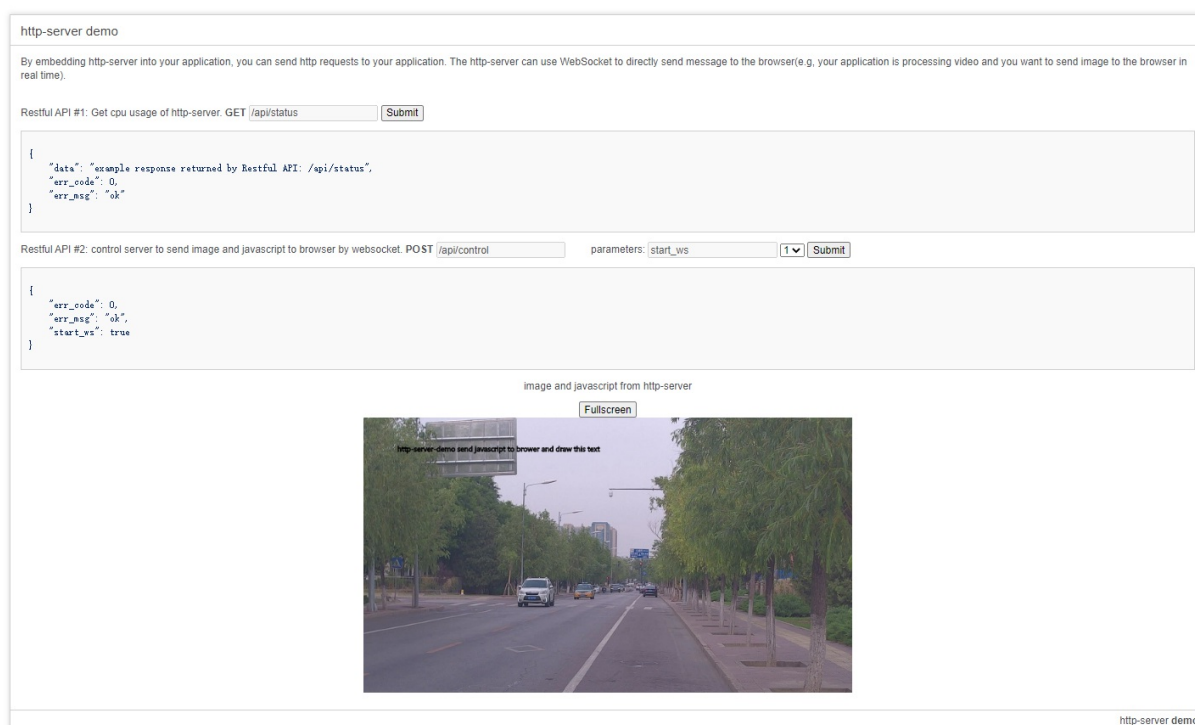
运行http-server-demo

在安装了SDK的开发机上终端执行：

1. `cd {SDK_INSTALL_DIR}/sysroots/aarch64-bst-linux/usr/bin/examples`
2. `adb push http-server-demo /usr/bin`
3. 运行adb shell进入A1000的shell环境，运行 `ifconfig` 获取A1000的ip地址；然后启动http-server-demo: `cd /usr/bin/http-server-demo && chmod +x http_server_demo && ./http_server_demo`

浏览器访问

打开浏览器，用 `http://{上面获取的A1000的ip}:8080`，得到一个网页如下图：



- Restful API #1: 点击submit，浏览器就会发起一个GET到 `/api/status` 的请求，以json格式返回并显示在下方。
- Restful API #2: 把start_ws的参数选中“1”，点击submit，浏览器会发起一个参数 `start_ws=1` 的POST到 `/api/control` 的请求并且建立到Http-Server端的WebSocket链接。Http-Server端收到请求后会以json格式返回并显示在下方，然后开始循环读取http-server-demo-pics目录下的图片通过WebSocket协议发送到浏览器端渲染出来。把start_ws的参数选中“0”，点击submit，浏览器会发起一个参数 `start_ws=0` 的POST到 `/api/control` 的请求，Http-Server端收到请求后会停止往浏览器端发送WebSocket数据。

- [local-visualizer-demo](#)(后续提供)

local-visualizer-demo(后续提供)

TODO

- [recorder-demo](#)
 - [查看录像文件详情:](#)
 - [播放录像文件:](#)
 - [录制录像文件:](#)

recorder-demo

运行cyber_recorder相关demo必须首先将cyber_recorder及测试文件demo.record拷贝到A1000指定路径:

```
cd {SDK_INSTALL_DIR}/sysroots/aarch64-bst-linux/usr/bin/cyber/tools/cyber_recorder
adb push cyber_recorder /usr/bin
cd {SDK_INSTALL_DIR}/sysroots/aarch64-bst-linux/usr/bin/examples/recorder-demo/record-data
adb push demo.record /usr/bin
```

查看录像文件详情:

```
$ cyber_recorder info demo.record
```

播放录像文件:

```
$ cyber_recorder play -f demo.record
```

录制录像文件:

在播放录像文件的同时, 另起一个终端执行以下命令, 就可以把收到的数据录制到一个以时间命名的新的record包中, **ctrl + C** 停止后, 在当前文件夹中查看是否生成新的record包。

```
$ cyber_recorder record -a
```

- [ros1_bridge-demo](#)
 - 环境搭建
 - 运行说明

ros1_bridge-demo

环境搭建

ros1环境设置

在一台机器上安装ros1，推荐按照ros1官网安装，网址为：<http://wiki.ros.org/melodic/Installation/Ubuntu>

对于ros1来说需要区分主机和从机，主机需要启动ros的master节点，另一个则为从机。欲两台机器进行通信，则先需要相互ping通。

- （一）在两台机器上分别查看ip及hostname，分别为主机ip、主机hostname、从机ip、从机hostname。（注意：主机和从机的hostname不能相同）
- （二）hosts文件设置
 - 在两台机器上分别打开hosts文件，命令为：`sudo vi /etc/hosts`
 - 在两台机器的hosts文件末端分别添加两台机器的ip和hostname

```
主机ip 主机hostname
从机ip 从机hostname
```

- 重启两台机器的网络设置，终端命令为：`sudo /etc/init.d/networking restart`(A1000上不需要执行)
- （三）在要启动ros1和bridge的节点的机器上进行设置：

若能找到bashrc文件

- 在终端打开bashrc文件，命令为：`sudo vi ~/.bashrc`
- 在主机的文件末尾添加的内容为

```
export ROS_HOSTNAME=主机hostname
export ROS_MASTER_URI=http://主机ip:11311
```

- 在从机的文件末尾添加的内容为

```
export ROS_HOSTNAME=从机hostname
export ROS_MASTER_URI=http://主机ip:11311
```

- 在所有启动ros1和bridge的终端上执行：`source ~/.bashrc`。若找不到bashrc文件，临时修改环境变量
- 在所有启动ros1和bridge的终端上,直接输入

```
export ROS_MASTER_URI=http://主机ip:11311
```

- （四）在主机上启动master节点和ros1的收发节点的终端需要找到devel/setup.bash进行source

BST-Platform环境设置

在启动BST-Platform和bridge节点的终端进行环境设置

```
source /usr/bin/cyber/setup.bash
```

运行说明

adb push所需lib和bin文件

以0.8.1版本为例，则所需要的库在/opt/bstos/0.8.1/sysroots/aarch64-bst-linux/usr/lib64，需要拷贝到A1000的/usr/lib64下

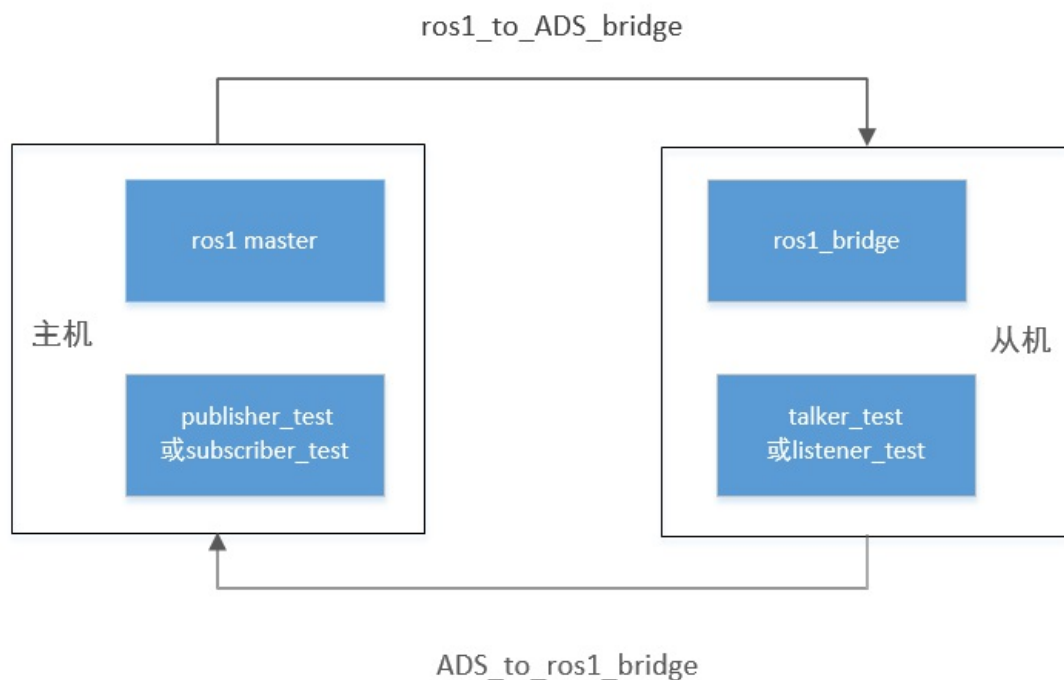
```
cd /opt/bstos/0.8.1/sysroots/aarch64-bst-linux/usr/lib64
adb push libcyber.so libcyber_proto.so libros_melodic.so /usr/lib64
```

则所需要ros1_bridge的bin文件在/opt/bstos/0.8.1/sysroots/aarch64-bst-linux/usr/bin/examples/ros1_bridge-demo/，需要拷贝到A1000的/usr/bin下

```
cd /opt/bstos/0.8.1/sysroots/aarch64-bst-linux/usr/bin/examples/ros1_bridge-demo/
adb push cyber_ros1_bridge-demo/* /usr/bin/
adb push cyber-demo/* /usr/bin
```

节点运行

为了开发者更方便的使用此模块，如下描述完整的使用逻辑。假设主机存在ros1 master节点和ros1的发送节点publisher_test、接收节点subscriber_test节点；从机上存在ros1_bridge节点和BST-Platform的发送节点talker_test、接收节点listener_test。使用ros1_bridge实现ros1与BST-Platform的互相通信，所以存在两种情况。一种情况是BST-Platform发送，ros1接收；另一种情况是ros1发送，BST-Platform接收。



- （一）BST-Platform发送，ros1接收时，需要启动四个终端。主机启动ros1 master节点、ros1接收节点；从机启动ADS_to_ros1_bridge节点、BST-Platform发送节点。

- (a)启动方法:

主机一个终端上启动ros1 master节点

```
roscore
```

主机另一个终端启动ros1接收节点

```
roslaunch test subscriber_test
```

从机一个终端启动ADS_to_ros1_bridge节点

```
./ADS_to_ros1_bridge
```

从机另一个终端启动BST-Platform发送节点

```
./talker_test 1000 10
```

注意事项: 1000为以字节为单位的数据大小, 10为发送频率

- (b)停止方法:

ctrl+c 或者 ctrl+z 或者 kill -9 进程号

- (c)结果判断:

若成功, 发送端 (BST-Platform发送节点) c和接收端 (ros1接收节点) 都有持续输出, 否则, 表明测试失败。

- (二) ros1发送, BST-Platform接收时, 需要启动四个终端。主机启动ros1 master节点、ros1发送节点; 从机启动ros1_to_ADS_bridge节点、BST-Platform接收节点。

- (a)启动方法:

主机一个终端上启动ros1 master节点

```
roscore
```

主机另一个终端启动ros1发送节点

```
roslaunch test publisher_test 1000 10
```

从机一个终端上启动ros1_to_ADS_bridge节点

```
./ros1_to_ADS_bridge
```

从机另一个终端启动BST-Platform接收节点

```
./listener_test
```

注意事项: 1000为以字节为单位的数据大小, 10为发送频率

- (b)停止方法:

ctrl+c 或者 ctrl+z 或者 kill -9 进程号

◦ (c)结果判断:

若成功, 发送端 (ros1发送节点) 和接收端 (BST-Platform接收节点) 都有持续输出, 否则, 表明测试失败。

注意事项

- ros1_bridge需要的ros1是基于melodic版本进行的裁减, 只包含通信相关的代码。所以ros1_bridge需要用户把.msg生成对应的.h拷贝到实际的工程中;
- 与ros1_bridge通信的ros1版本推荐使用比较新的版本 (如 melodic版本) 或者使用比较广泛的版本 (如 indigo版本);
- ros1_bridge所需的ros1 message头文件需与ros1发送或者接收节点message头文件一致;
- ros1_bridge所需的BST-Platform message头文件与BST-Platform发送或者接收节点message头文件一致;

- [ros2_bridge-demo](#)
 - [ros2 环境测试](#)
 - [PC机平台ROS2环境](#)
 - [A1000 平台ROS2环境](#)
 - [执行ROS2桥例程](#)

ros2_bridge-demo

这是一个使用ros2-bridge的简单示例。

ros2 环境测试

ros2-bridge示例运行需要两个ROS2的环境：一个只有ROS2环境，一个ROS2通信功能+cyber环境。运行前需要先测试这两个环境是否搭建好。

PC机平台ROS2环境

- 在一台机器上安装ros2，推荐按照ROS2官网安装。（网址为：<https://index.ros.org/doc/ros2/Installation/Dashing/>）"Building from source"进行安装并测试。
- 按照下面修改ros2_dashing/src/ros2/demos/demo_nodes_cpp/src/topics文件夹里相关文件

```
listener.cpp : 修改 auto topic = std::string("chatter"); 为
auto topic = std::string("rosnode");

talker.cpp: 修改 auto topic = std::string("chatter"); 为
auto topic = std::string("rosnode1");
```

- 一个终端运行 `ros2 run demo_nodes_cpp talker`；另一个终端运行 `ros2 run demo_nodes_cpp listener`

A1000 平台ROS2环境

adb push所需lib和bin文件

系统打包时会做这部分工作，但因为ROS2-bridge不是必须项，如果没有打包进系统，测试需要测按下面步骤操作

（一）以0.8.1版本为例，则所需要的库在/opt/bstos/0.8.1/sysroots/aarch64-bst-linux/usr/lib64，需要拷贝到A1000的/usr/lib64下

```
cd /opt/bstos/0.8.1/sysroots/aarch64-bst-linux/usr/lib64
adb push libddsc* libyaml* librmw* /usr/lib64
adb push librc1* libros* libtype* /usr/lib64
adb push libmypoco.so librcutils.so libunique* libbuilt* /usr/lib64
adb push liblifecyc* libcompos* /usr/lib64
adb push libaction_msgs.so libstd_msgs.so /usr/lib64
```

（二）所需要ros2_bridge的bin文件在/opt/bstos/0.8.1/sysroots/aarch64-bst-linux/usr/bin/examples/ros2_bridge-demo/，需要拷贝到A1000的/usr/bin下

```
cd /opt/bstos/0.8.1/sysroots/aarch64-bst-linux/usr/bin/examples/ros2_bridge-demo/
adb push cyber_ros2_bridge-demo/* /usr/bin/examples/ros2_bridge-demo/cyber_ros2_bridge-demo
```

运行bin文件

- （一）打开三个终端A /B /C 分别依次执行以下命令：

注： 如果bridge进程和cyber的进程位于同一块板子或者同一主机时，跳过以下步骤；当bridge进程和cyber的进程位于不同板子或者主机时需要执行以下步骤。

- `cd /usr/bin/`
- `ifconfig` （查看ip 地址）
- `vi cyber/setup.bash` (修改`export CYBER_IP=127.0.0.1` 为查看到的地址比如`export CYBER_IP=192.168.2.55`)
- `source cyber/setup.bash`

- （二）运行listen_ros

终端A 运行

```
cd /usr/bin/examples/ros2_bridge-demo/cyber_ros2_bridge-demo
./listen_ros rosnode1
```

判断依据： 屏幕打印出与PC机平台运行 `ros2 run demo_nodes_cpp talker` 输出一致说明ROS2 在A1000 平台接收正确

`ctrl+c` 终止运行 `listen_ros`

保持终端A

- （三）talk_ros

终端B 运行

```
cd /usr/bin/examples/ros2_bridge-demo/cyber_ros2_bridge-demo
./talk_ros rosnode
```

判断依据： 屏幕输出与PC机平台运行 `ros2 run demo_nodes_cpp listener` 输出一致说明ROS2 在A1000 平台发送正确

`ctrl+c` 终止运行 `talk_ros`

保持终端B

`ctrl+c` 终止运行 PC机平台的talker和listener两终端

执行ROS2桥例程

1 启动PC机平台的listener终端 :`ros2 run demo_nodes_cpp listener`

2 启动PC机平台的talker终端 :`ros2 run demo_nodes_cpp talker`

3 终端A 运行

```
./multibridge
```

4 终端B 运行

```
./cyber_listener cybernode1
```

判断依据：通信正常该终端应打印与ROS2主机A talker 发送一样的信息(会丢失第一帧数据属通信正常) 注：步骤2~4最好现在各个终端先输好命令，接着依次快速回车终端，只会丢失第一帧数据（这个无法避免，ros2与cyber在服务发现需要时间，可以采用重发第一帧解决。ros2_demo只是个简单测试桥收发，所以实例代码并没有将第一帧发两次）

5 终端C 运行

```
cd /usr/bin/examples/ros2_bridge-demo/cyber_ros2_bridge-demo  
./cyber_talker
```

判断依据：通信正常ROS2主机A listener应打印与 该终端发送一样的信息 (会丢失第一帧数据属通信正常)

- [transform-demo](#)
 - [拷贝transform-demo示例至A1000](#)
 - [拷贝camera-fusion-demo示例至A1000](#)
 - [在A1000上运行transform-demo示例](#)
 - [在A1000上运行camera-fusion-demo示例](#)
 - [在A1000上播放record](#)

transform-demo

说明:

1. transform测试用例不能单独使用，需与sensor-fusion或者其他需要坐标转换功能的模块联合使用。
2. transform测试用例只实现静态坐标转换发布，动态坐标转换发布和坐标转换应用两大功能未实现，动态坐标转换发布用于定位模块发布车辆坐标系到世界坐标系的转换关系，当前还未实现定位模块，故暂时不提供测试用例；坐标转换应用用于接收保存静态坐标转换发布和动态坐标转换发布两功能发布的转换关系，一般作为API，被其他模块调用。在camera-fusion和radar-fusion已实现调用。

transform测试用例使用如下:

拷贝transform-demo示例至A1000

本demo已经集成诊断功能，拷贝诊断配置表至A1000对应目录（若诊断配置表已经拷贝过，可忽略本步骤操作）:

```
cd /opt/bstos/XXXX/sysroots/x86_64-bstsdk-linux/usr/bin/native-tools/diagnostic_conversion/output
adb push diagnostic_did.bin /etc/diagnostic_bin
adb push diagnostic_dtc.bin /etc/diagnostic_bin
```

注意：每次push diagnostic_did.bin和diagnostic_dtc.bin完成后，需重启A1000以覆盖原来文件。

运行transform-demo前，需要将transform-demo示例拷贝至A1000对应目录:

```
cd /opt/bstos/XXXX/sysroots/aarch64-bst-linux/usr/bin/examples
adb shell mkdir -p /usr/bin/examples
adb push transform-demo /usr/bin/examples
cd /opt/bstos/XXXX/sysroots/aarch64-bst-linux/usr/lib64
adb push libtransform_statictransform_example.so /usr/lib64
adb push libtransform_statictransform_proto.so /usr/lib64
```

切换到A1000该目录,为程序添加可执行权限:

```
运行adb shell进入a1000的shell环境，运行`cd /usr/bin/examples/transform-demo && chmod +x *`
```

拷贝camera-fusion-demo示例至A1000

运行transform-demo前，将需要的示例拷贝至A1000对应目录:

```
cd /opt/bstos/XXXX/sysroots/aarch64-bst-linux/usr/bin/examples/sensorfusion-demo
adb shell mkdir -p /usr/bin/examples/sensorfusion-demo
adb push camera-fusion-demo /usr/bin/examples/sensorfusion-demo
adb push radar-fusion-demo /usr/bin/examples/sensorfusion-demo
cd /opt/bstos/XXXX/sysroots/aarch64-bst-linux/usr/lib64
adb push libsensordfusion_camerafusion_example.so /usr/lib64
adb push libsensordfusion_camerafusion_proto.so /usr/lib64
```

切换到A1000该目录,为程序添加可执行权限:

```
运行adb shell进入a1000的shell环境
cd /usr/bin/examples/sensorfusion-demo/camera-fusion-demo && chmod +x *
```

在A1000上运行transform-demo示例

新建一个终端1,依次执行以下步骤:

```
运行adb shell进入a1000的shell环境
cd /usr/bin/examples/transform-demo/dag
source /usr/bin/cyber/setup.bash
mainboard -d ./example_transform_statictransform.dag
```

终端输出信息:

终端1将会输出以下信息:

```
INFO: []command: mainboard -d ./example_transform_statictransform.dag INFO: []host ip: 192.168.8.71[Global Data]
[Global Data] cyber config file: /usr/local/bin/cyber/conf/cyber.pb.conf
[Global Data] current process_group_is:mainboard_11773
binary_name_is mainboard, process_group_is mainboard_default, has 1 dag conf
dag_conf: ./example_transform_statictransform.dag
INFO: [mainboard]Register exit handle succ.INFO: [mainboard]Start initialize dag: /usr/local/bin/examples/transform-demo/dag/./example_transform_statictransform.dagINFO: [mainboard]Begin LoadLibrary: /usr/local/lib/libtransform_statictransform_example.so[Scheduler]expected cfg_file of this group is /usr/local/bin/cyber/conf/mainboard_default.conf
INFO: [mainboard]Broadcast static transform, frame id [novatel], child frame id [velodyne128]
INFO: [mainboard]Broadcast static transform, frame id [world], child frame id [novatel]
INFO: [mainboard]Broadcast static transform, frame id [velodyne128], child frame id [front_6mm]
INFO: [mainboard]Broadcast static transform, frame id [velodyne128], child frame id [front_12mm]
INFO: [mainboard]Broadcast static transform, frame id [velodyne128], child frame id [radar_front]
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task0
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task1
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task2
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task3
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task4
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task5
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task6
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task7
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task8
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task9
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task10
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task11
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task12
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task13
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task14
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task15
```

表明transform模块发布了不同坐标系之间的转换关系。

在A1000上运行camera-fusion-demo示例

新建一个终端2,依次执行以下步骤:

```
运行adb shell进入a1000的shell环境
cd /usr/bin/examples/sensorfusion-demo/camera-fusion-demo
source /usr/bin/cyber/setup.bash

mainboard -d ./example_camera_fusion.dag
```

终端输出信息: 此时,终端2将会输出以下信息:


```
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task0
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task1
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task2
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task3
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task4
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task5
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task6
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task7
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task8
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task9
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task10
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task11
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task12
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task13
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task14
INFO: [mainboard]Scheduler::CreateTask create croutine: /internal/task15
INFO: [mainboard]receive transform msgs !
```

表明camera-fusion模块成功接收到了transform发布的信息。

在A1000上播放record

新建一个终端3，依次执行以下步骤：

```
运行adb shell进入a1000的shell环境
source /usr/bin/cyber/setup.bash
cd /usr/bin/cyber/tools/cyber_recorder
./cyber_recorder play -f \
/usr/bin/examples/sensorfusion-demo/camera-fusion-demo/test-data/camerafusion-demo-test-data.record
```

终端切换到终端2，可以看到终端上面不断打印出transform信息（如下图所示），表明camera-fusion不断调用transform坐标转换信息，成功实现坐标转换：

```
[TRANSFORM] Get pose timestamp: 1.51381e+09, pose:
 0.998235 0.00317619 -0.0593108 -0.27913
 0.0593826 -0.0323602 0.99771 2.013
 0.00124955 -0.99947 -0.0324917 -0.223188
      0      0      0      1
```


- **transform_util_demo**
 - 拷贝transform_util_demo示例至A1000
 - 在A1000上运行transform_util_demo示例

transform_util_demo

transform_util_demo的使用如下:

拷贝transform_util_demo示例至A1000

运行transform_util_demo前, 需要将transform_util_demo示例拷贝至A1000对应目录:

```
cd /opt/bstos/XXXX/sysroots/aarch64-bst-linux/usr/bin/examples
adb shell mkdir -p /usr/bin/examples
adb push transform-demo /usr/bin/examples
cd /opt/bstos/XXXX/sysroots/aarch64-bst-linux/usr/lib64
adb push libtransform_util.so /usr/lib64
```

切换到A1000该目录,为程序添加可执行权限:

```
运行adb shell进入a1000的shell环境, 运行`cd /usr/bin/examples/transform-demo/transform_util_demo && chmod +x *`
```

在A1000上运行transform_util_demo示例

新建一个终端1, 依次执行以下步骤:

```
运行adb shell进入a1000的shell环境
cd /usr/bin/examples/transform-demo/transform_util_demo
source /usr/bin/cyber/setup.bash
./transform_util_demo
```

根据终端信息, 输入要转换的坐标, 终端会自动打印相关的转换结果。各提示信息解释如下:

- please enter Ego longitude in degrees [ie. 116.39123343289631]: // 输入自车位置的经度信息 (WGS84坐标系), 以度为单位;
- please enter Ego latitude in degrees [ie. 39.9072885060602]: // 输入车辆自车位置的纬度信息 (WGS84坐标系), 以度为单位;
- please enter Ego altitude in m [ie. 0.0]: // 输入车辆自车位置的海拔高度 (WGS84坐标系), 以米为单位;
- EGO GCJ02 --> 104.052750051690623 30.6976491773963609 505 // 根据输入的WGS84坐标, 得到的GCJ02火星坐标系下的坐标;
- EGO WebMercator --> 11620778.949031774 3620714.63523430331 470 // 根据输入的WGS84坐标, 得到的Web Mercator投影坐标系下的坐标;
- please enter Ego utm zone [ie. 43]: // 输入要进行计算的UTM区域编号;
- EGO UTM --> east:3338078.54328177217 north:3784730.871157235 zone:43 // 根据输入的WGS84坐标和UTM区域, 得到UTM坐标;
- please enter Target longitude in degrees [ie. 116.39123343289631]: // 输入目标车辆位置的经度信息 (WGS84坐标系), 以度为单位;
- please enter Target latitude in degrees [ie. 39.9072885060602]: // 输入目标车辆位置的纬度信息 (WGS84坐标系), 以度为单位;
- please enter Target altitude in m [ie. 0.0]: // 输入目标车辆位置的海拔高度 (WGS84坐标系), 以米为单位

- TARGET ENU --> East:0.00443561048110696632 North:30.7980945725565221
Up:-7.46575694012818758e-05 // 根据输入的自车经纬度、目标车辆的经纬度，转换得到的目标车辆ENU坐标系信息，ENU坐标系以自车位置为原点。
- please enter Target East Position in meters [ie. 10]: // 输入目标车辆在自车ENU坐标系下的东向位置，以米为单位；
- please enter Target North Position in meters [ie. 10]: // 输入目标车辆在自车ENU坐标系下的北向位置，以米为单位；
- please enter Target Up Position in meters [ie. 0.0]: // 输入目标车辆在自车ENU坐标系下的天线位置，以米为单位；
- TARGET WGS84 --> 104.05027527777786 30.6997269481377408 505.000047822482884 // 根据目标车辆的ENU位置和自车经纬度，转换得到的目标车辆经纬度信息。

```
=====TRANSFORM UTIL TEST=====
please enter Ego longitude in degrees [ie. 116.39123343289631]:
104.0502752777778
please enter Ego latitude in degrees [ie. 39.9072885060602]:
30.7000787037037
please enter Ego altitude in m [ie. 0.0]:
505
EGO GCJ02 --> 104.052750051690623 30.6976491773963609 505
please enter Ego utm zone [ie. 43]:
43
EGO UTM --> east:3338078.54328177217 north:3784730.871157235 zone:43
please enter Target longitude in degrees [ie. 116.39123343289631]:
104.0502753240741
please enter Target latitude in degrees [ie. 39.9072885060602]:
30.70035648148148
please enter Target altitude in m [ie. 0.0]:
505
TARGET ENU --> East:0.00443561048110696632 North:30.7980945725565221 Up:-7.46575694012818758e-05
```

本demo会循环测试，因此可以持续进行测试。

退出办法：ctrl + C后，Ctrl + Z。

- [camera-fusion-demo](#)
 - [拷贝示例至A1000](#)
 - [在A1000上运行示例](#)
 - [单通道输入](#)
 - [双通道输入](#)

camera-fusion-demo

camera-fusion-demo位于/opt/bstos/XXXX/sysroots/aarch64-bst-linux/usr/bin/examples/sensorfusion-demo/camera-fusion-demo目录下。

camera fusion测试数据为camerafusion-demo-test-data.record和camerafusion-long-short-2-channel-test-data.record，在camera-fusion-demo/test-data目录下。

拷贝示例至A1000

1.本demo已经集成诊断功能，拷贝诊断配置表至A1000对应目录（若诊断配置表已经拷贝过，可忽略本步骤操作）：

```
cd /opt/bstos/XXXX/sysroots/x86_64-bstsdk-linux/usr/bin/native-tools/diagnostic_conversion/output
adb push diagnostic_did.bin /etc/diagnostic_bin
adb push diagnostic_dtc.bin /etc/diagnostic_bin
```

注意：每次push diagnostic_did.bin和diagnostic_dtc.bin完成后，需重启A1000以覆盖原来文件。

2.运行camera-fusion-demo前，将需要的示例拷贝至A1000对应目录：

```
cd /opt/bstos/XXXX/sysroots/aarch64-bst-linux/usr/bin/examples/sensorfusion-demo
adb shell mkdir -p /usr/bin/examples/sensorfusion-demo
adb push camera-fusion-demo /usr/bin/examples/sensorfusion-demo
adb push radar-fusion-demo /usr/bin/examples/sensorfusion-demo
cd /opt/bstos/XXXX/sysroots/aarch64-bst-linux/usr/lib64
adb push libsensofusion_camerafusion_example.so /usr/lib64
adb push libsensofusion_camerafusion_proto.so /usr/lib64
adb push libtransform_statictransform_example.so /usr/lib64
adb push libtransform_statictransform_proto.so /usr/lib64
```

3.切换到A1000该目录,为程序添加可执行权限：

```
运行adb shell进入a1000的shell环境
cd /usr/bin/examples/sensorfusion-demo/camera-fusion-demo && chmod +X *
```

在A1000上运行示例

单通道输入

1.确保诊断服务启动,查看诊断服务状态：

```
运行adb shell进入a1000的shell环境
systemctl status diagnose-service
```

```

[0;1;32m[0m diagnose-service.service - DiagnoseService systemd service.
Loaded: loaded (/lib/systemd/system/diagnose-service.service; disabled; vendor preset: enabl
ed)
Active: [0;1;32mactive (running)[0m since Thu 2020-11-19 02:32:17 UTC; 7min ago
Main PID: 413 (DiagnoseService)
Tasks: 8 (limit: 2152)
Memory: 5.7M
CGroup: /system.slice/diagnose-service.service
└─413 /usr/bin/DiagnoseService

```

2.新建一个终端1，依次执行以下步骤：

```

运行adb shell进入a1000的shell环境
cd /usr/bin/examples/transform-demo/dag
source /usr/bin/cyber/setup.bash
mainboard -d ./example_transform_statictransform.dag

```

3.新建一个终端2，依次执行以下步骤：

```

运行adb shell进入a1000的shell环境
cd /usr/bin/examples/sensorfusion-demo/camera-fusion-demo
source /usr/bin/cyber/setup.bash
VSOMEIP_CONFIGURATION=/etc/bstos_service/diagnosticservice/vsomeip-diag.json \
mainboard -d ./example_camera_fusion.dag 启动camera fusion程序

```

4.新建一个终端3，依次执行以下步骤：

```

运行adb shell进入a1000的shell环境
source /usr/bin/cyber/setup.bash
cd /usr/bin/cyber/tools/cyber_recorder
./cyber_recorder play -f \
/usr/bin/examples/sensorfusion-demo/camera-fusion-demo/test-data/camerafusion-demo-test-data.record

```

5.此时终端2开始camera fusion目标融合处理，终端窗口不断打印出处理结果及耗时情况。

双通道输入

1.确保诊断服务启动,查看诊断服务状态：

```

运行adb shell进入a1000的shell环境
systemctl status diagnose-service

```

```

[0;1;32m[0m diagnose-service.service - DiagnoseService systemd service.
Loaded: loaded (/lib/systemd/system/diagnose-service.service; disabled; vendor preset: enabl
ed)
Active: [0;1;32mactive (running)[0m since Thu 2020-11-19 02:32:17 UTC; 7min ago
Main PID: 413 (DiagnoseService)
Tasks: 8 (limit: 2152)
Memory: 5.7M
CGroup: /system.slice/diagnose-service.service
└─413 /usr/bin/DiagnoseService

```

2.新建一个终端1，依次执行以下步骤：

```

运行adb shell进入a1000的shell环境
cd /usr/bin/examples/transform-demo/dag
source /usr/bin/cyber/setup.bash
mainboard -d ./example_transform_statictransform.dag

```

3.新建一个终端2，依次执行以下步骤：

```
运行adb shell进入a1000的shell环境
cd /usr/bin/examples/sensorfusion-demo/camera-fusion-demo
source /usr/bin/cyber/setup.bash
VSOMEIP_CONFIGURATION=/etc/bstos_service/diagnosticservice/vsomeip-diag.json \
mainboard -d ./example_camera_fusion.dag 启动camera fusion程序
```

4.新建一个终端3，依次执行以下步骤：

```
运行adb shell进入a1000的shell环境
source /usr/bin/cyber/setup.bash
cd /usr/bin/cyber/tools/cyber_recorder
./cyber_recorder play -f \
/usr/bin/examples/sensorfusion-demo/camera-fusion-demo/test-data/camerafusion-long-short-2-channel-test-data.record
```

5.此时终端2开始camera fusion目标融合处理，终端窗口不断打印出双通道输入数据处理结果及耗时情况。

- [radar-fusion-demo](#)
 - [拷贝示例至A1000](#)
 - [在A1000上运行示例](#)

radar-fusion-demo

radar-fusion-demo位于/opt/bstos/XXXX/sysroots/aarch64-bst-linux/usr/bin/examples/sensorfusion-demo/radar-fusion-demo目录下。

radar fusion测试数据为radarfusion-demo-test-data.record，在radar-fusion-demo/test-data目录下。

拷贝示例至A1000

1.本demo已经集成诊断功能，需拷贝诊断配置表至A1000对应目录（若诊断配置表已经拷贝过，可忽略本步骤操作）：

```
cd /opt/bstos/XXXX/sysroots/x86_64-bstsdk-linux/usr/bin/native-tools/diagnostic_conversion/output
adb push diagnostic_did.bin /etc/diagnostic_bin
adb push diagnostic_dtc.bin /etc/diagnostic_bin
```

注意：每次push diagnostic_did.bin和diagnostic_dtc.bin完成后，需重启A1000以覆盖原来文件。

2.运行radar-fusion-demo前，将需要的示例拷贝至A1000对应目录：

```
cd /opt/bstos/XXXX/sysroots/aarch64-bst-linux/usr/bin/examples/sensorfusion-demo
adb shell mkdir -p /usr/bin/examples/sensorfusion-demo
adb push camera-fusion-demo /usr/bin/examples/sensorfusion-demo
adb push radar-fusion-demo /usr/bin/examples/sensorfusion-demo
cd /opt/bstos/XXXX/sysroots/aarch64-bst-linux/usr/bin/examples
adb push transform-demo /usr/bin/examples
cd /opt/bstos/XXXX/sysroots/aarch64-bst-linux/usr/lib64
adb push libsensordfusion_radarfusion_example.so /usr/lib64
adb push libsensordfusion_radarfusion_proto.so /usr/lib64
adb push libtransform_statictransform_example.so /usr/lib64
adb push libtransform_statictransform_proto.so /usr/lib64
```

3.切换到A1000该目录,为程序添加可执行权限：

```
运行adb shell进入a1000的shell环境
cd /usr/bin/examples/sensorfusion-demo/radar-fusion-demo && chmod +x *
cd /usr/bin/examples/transform-demo && chmod +x *
```

在A1000上运行示例

1.确保诊断服务启动,查看诊断服务状态：

```
运行adb shell进入a1000的shell环境
systemctl status diagnose-service
```

```
[[0;1;32m[[0m diagnose-service.service - DiagnoseService systemd service.
Loaded: loaded (/lib/systemd/system/diagnose-service.service; disabled; vendor preset: enabl
ed)
Active: [[0;1;32mactive (running) [[0m since Thu 2020-11-19 02:32:17 UTC; 7min ago
Main PID: 413 (DiagnoseService)
Tasks: 8 (limit: 2152)
Memory: 5.7M
CGroup: /system.slice/diagnose-service.service
└─413 /usr/bin/DiagnoseService
```

2.新建一个终端1，依次执行以下步骤：

```
运行adb shell进入a1000的shell环境
cd /usr/bin/examples/transform-demo/dag
source /usr/bin/cyber/setup.bash
mainboard -d ./example_transform_statictransform.dag
```

3.新建一个终端2，依次执行以下步骤：

```
运行adb shell进入a1000的shell环境
cd /usr/bin/examples/sensorfusion-demo/radar-fusion-demo/dag
source /usr/bin/cyber/setup.bash
VSOMEIP_CONFIGURATION=/etc/bstos_service/diagnosticservice/vsomeip-diag.json \
mainboard -d ./sensorfusion_radar_example_component.dag 启动radar fusion程序。
```

4.新建一个终端3，依次执行以下步骤：

```
运行adb shell进入a1000的shell环境
source /usr/bin/cyber/setup.bash
cd /usr/bin/cyber/tools/cyber_recorder
./cyber_recorder play -f \
/usr/bin/examples/sensorfusion-demo/radar-fusion-demo/test-data/radarfusion-demo-test-data.record
```

5.此时终端2开始radar fusion目标融合处理，终端窗口不断打印出处理结果及耗时情况。

- [adstime-demo](#)
 - [时间戳接口测试](#)

adstime-demo

目前，**adstime**只提供了将系统时间转换为时间戳的简单接口，之后会随着时间同步功能的开发进度，逐渐增加时间同步的演示。

切换到该目录：

```
cd /usr/bin/examples/autoplt-demo
```

时间戳接口测试

- 启动方法

启动一个终端：

在终端输入 `./example_adstime`

- 停止方法

ctrl + C

- 结果判断

若成功，在控制台会间隔**1s**输出一当前秒数时间戳、纳秒时间戳和日期字符串

- [task-manage-demo](#)(暂不支持)

task-manage-demo(暂不支持)

启动一个终端执行如下命令：

1. `cd /opt/bstos/0.8.1/sysroots/aarch64-bst-linux/usr/bin/examples`
2. `adb push task-manage-demo /usr/bin/`
3. `adb shell`
4. `cd /usr/bin/task-manage-demo`
5. `./task-manage-demo conf/sample.conf`

- [communicaton-safety-demo](#)
 - [E2E主机间通信测试](#)

communicaton-safety-demo

运行autopl-t-demo前，需要将示例和配置文件拷贝至A1000对应目录，如果A1000的/usr/bin/下没有examples目录，则需要创建。

以0.8.1版本为例：

```
scp -r /opt/bstos/0.8.1/sysroots/aarch64-bst-linux/usr/bin/examples/vsomeip-e2e-demo \
root@A1000_IP:/usr/bin/examples
```

切换到该目录

```
cd /usr/bin/examples/vsomeip-e2e-demo/
```

E2E主机间通信测试

注意两个A1000不能设置成固定IP

PROFILE01测试

启动方法

- （一）在A1000上启动一个终端，在终端输入如下命令：

```
env VSOMEIP_CONFIGURATION=e2e_test_client_external.json.in \
VSOMEIP_APPLICATION_NAME=client-sample-e2e ./e2e_test_client
```

注意：需要修改e2e_test_client_external.json.in中"unicast"变量为A1000的IP,假设A1000的IP为192.168.2.100，则

```
"unicast" : "192.168.2.100"
```

- （二）在另一个A1000上启动一个终端，在终端输入如下命令：

```
env VSOMEIP_CONFIGURATION=e2e_test_service_external.json.in \
VSOMEIP_APPLICATION_NAME=service-sample-e2e ./e2e_test_service
```

注意：需要修改e2e_test_service_external.json.in中"unicast"变量为A1000的IP,假设A1000的IP为192.168.2.101，则

```
"unicast" : "192.168.2.101"
```

停止方法

ctrl + C

结果判断

- （一）若成功：

在启动的client中终端会有10次E2E结果输出

第一次输出：

```
[info] E2E protection result is E2E_P_WRONGSEQUENCE for service: 1234 method: 8001
```

其余九次输出：

```
[info] E2E protection result is E2E_P_OK for service: 1234 method: 8001
```

在启动的service终端打印10次E2E输入的数据

```
[info] E2E input data:
(12) 00 00 00 ff ff ff ff ff ff ff ff
[info] E2E input data:
(12) 00 00 01 ff ff ff ff ff ff ff ff
[info] E2E input data:
(12) 00 00 02 ff ff ff ff ff ff ff ff
[info] E2E input data:
(12) 00 00 03 ff ff ff ff ff ff ff ff
[info] E2E input data:
(12) 00 00 04 ff ff ff ff ff ff ff ff
[info] E2E input data:
(12) 00 00 05 ff ff ff ff ff ff ff ff
[info] E2E input data:
(12) 00 00 06 ff ff ff ff ff ff ff ff
[info] E2E input data:
(12) 00 00 07 ff ff ff ff ff ff ff ff
[info] E2E input data:
(12) 00 00 08 ff ff ff ff ff ff ff ff
[info] E2E input data:
(12) 00 00 09 ff ff ff ff ff ff ff ff
```

- （二）若失败：

则没有上述结果输出。

PROFILE04测试

启动方法

- （一）在A1000上把e2e_test_client_external.json.in文件中修改为A1000的ip；且替换关于e2e的部分，要替换的内容如下：

```
"e2e" :
{
  "e2e_enabled" : "true",
  "protected" :
  [
    {
      "service_id" : "0x1234",
      "event_id" : "0x8001",
      "profile" : "PROFILE04",
      "variant" : "checker",
      "data_id" : "0xA73"
    }
  ]
},
```

在A1000上启动一个终端，在终端输入如下命令：

```
env VSOMEIP_CONFIGURATION=e2e_test_client_external.json.in \
VSOMEIP_APPLICATION_NAME=client-sample-e2e ./e2e_test_client
```

- （二）在另一个A1000上把e2e_test_service_external.json.in文件中修改为A1000的ip；且替换关于e2e的部分，要替换的内容如下：

```
"e2e" :
{
  "e2e_enabled" : "true",
  "protected" :
  [
    {
      "service_id" : "0x1234",
      "event_id" : "0x8001",
      "profile" : "PROFILE04",
      "variant" : "protector",
      "data_id" : "0xA73"
    }
  ]
},
```

在此A1000上启动一个终端，在终端输入如下命令：

```
env VSOMEIP_CONFIGURATION=e2e_test_service_external.json.in \
VSOMEIP_APPLICATION_NAME=service-sample-e2e ./e2e_test_service
```

停止方法

ctrl + C

结果判断

- （一）若成功：

在启动的client中终端会有10次E2E结果输出

```
[info] E2E protection result is E2E_P_OK for service: 1234 method: 8001
```

在启动的service终端打印10次E2E输入的数据

```
[info] E2E input data:
(12) 00 00 00 ff ff ff ff ff ff ff ff
[info] E2E input data:
(12) 00 00 01 ff ff ff ff ff ff ff ff
[info] E2E input data:
(12) 00 00 02 ff ff ff ff ff ff ff ff
[info] E2E input data:
(12) 00 00 03 ff ff ff ff ff ff ff ff
[info] E2E input data:
(12) 00 00 04 ff ff ff ff ff ff ff ff
[info] E2E input data:
(12) 00 00 05 ff ff ff ff ff ff ff ff
[info] E2E input data:
(12) 00 00 06 ff ff ff ff ff ff ff ff
[info] E2E input data:
(12) 00 00 07 ff ff ff ff ff ff ff ff
[info] E2E input data:
(12) 00 00 08 ff ff ff ff ff ff ff ff
[info] E2E input data:
```

```
(12) 00 00 09 ff ff ff ff ff ff ff ff
```

- （二）若失败：

则没有上述结果输出。

- [diagnostic-demo](#)
 - [拷贝示例至A1000](#)
 - [运行示例](#)

diagnostic-demo

diagnostic-demo提供diagnostic接口使用示例。

diagnostic-demo位于/opt/bstos/XXXX/sysroots/aarch64-bst-linux/usr/bin/examples/diagnostic-demo目录下。

拷贝示例至A1000

运行diagnostic-demo前，需要将示例拷贝至A1000对应目录：

```
cd /opt/bstos/XXXX/sysroots/x86_64-bstsdk-linux/usr/bin/native-tools/diagnostic_conversion/output
adb push diagnostic_did.bin /etc/diagnostic_bin
adb push diagnostic_dtc.bin /etc/diagnostic_bin
cd /opt/bstos/XXXX/sysroots/aarch64-bst-linux/usr/bin/examples
adb shell mkdir -p /usr/bin/examples
adb push diagnostic-demo /usr/bin/examples
cd /opt/bstos/XXXX/sysroots/aarch64-bst-linux/usr/lib64
adb push libdiagnostic_example.so /usr/lib64
adb push libexample_diagnostic_conf_proto.so /usr/lib64
```

切换到A1000该目录,为程序添加可执行权限：

```
运行adb shell进入a1000的shell环境，运行`cd /usr/bin/examples/diagnostic-demo && chmod +X *`
```

注意：每次push diagnostic_did.bin和diagnostic_dtc.bin完成后，需重启以覆盖原来文件。

运行示例

1. TC397端：使用串口工具连接TC397
2. TC397端：输入ifconfig查询IP是否为192.168.10.8，若不是，使用以下命令修改IP

```
ifconfig eth "192.168.10.8"
```

3. TC397端：使用CAN盒子连接TC397的CAN1，并打开CANTest软件（默认使用CANFD，配置为1M+5M）
4. A1000端：使用以下命令修改A1000 IP地址

```
systemctl stop NetworkManager
ifconfig eth0 0
ifconfig eth0 192.168.10.xx(1~255自选，不能与TC397相同)
ifconfig eth0 up
ping 192.168.10.8(确认是否能与TC397通信)
```

5. A1000端：新建一个终端，依次执行以下步骤：

```
cd /usr/bin/examples/diagnostic-demo/dag
source /usr/bin/cyber/setup.bash
VSOMEIP_CONFIGURATION=/etc/bstos_service/diagnosticservice/vsomeip-diag.json \
mainboard -d ./example_diagnostic_component.dag
```

6. TC397端：使用以下命令打开打印

```
ethprintf enable
```

7. TC397端：使用CANTest发送DID诊断数据

- 1) 03 22 xx xx(读取DID信息)
- 2) 03 2e xx xx(写入DID信息)
- 3) 03 2f xx xx(输入输出IO控制)
- 4) 04 14 FF FF FF(清除所有dtc)
- 5) 03 19 01 FF(读取所有dtc)

8. TC397端：在串口打印窗口查看A1000返回的DID数据

- **timed**
 - 配置文件
- This is the most basic ntp configuration file
- The driftfile must remain in a place specific to this
- machine - it records the machine specific clock error
- This should be a server that is close (in IP terms)
- to the machine. Add other servers as required.
- Unless you un-comment the line below ntpd will sync
- only against the local system clock.
- server time.server.example.com
- Using local hardware clock as fallback
- Disable this when using ntpd -q -g -x as ntpdate or it will sync to itself
- Defining a default security setting

timed

timed是用于配置和管理时间同步守护进程。用户通过配置文件，可以打开或关闭指定的时间同步功能，并设置必要的参数。**timed**的配置中，可以同时配置为NTP_SERVER和PTP_SERVER，但GNSS、PTP_CLIENT和NTP_CLIENT只能选择其中一项，当选择多个时，配置文件解析按照GNSS > PTP_CLIENT > NTP_CLIENT的优先级，选择优先级最高的进行配置。**timed**配置文件格式为：

```

syntax = "proto2";

package autoplt.timed.config;

enum GnssReceiverPortType{
    // UART0默认为A55-debug口，不允许将其配置为GNSS接收机端口
    // UART1默认为Safety-debug口，不允许将其配置为GNSS接收机端口
    UART2 = 3;
    UART3 = 4;
    SPI0 = 5;
    SPI1 = 6;
    SPI2 = 7;
    SPI3 = 8;
    IIC0 = 9;
    IIC1 = 10;
    IIC2 = 11;
    IIC3 = 12;
    ETH0 = 13;
    ETH1 = 14;
    ETH2 = 15;
    CAN0 = 16;
    CAN1 = 17;
};

// 目前支持两种接收机型号
enum GnssReceiverModel{
    UBLOX_ZEDF9K = 2;
    BYNAV_BY682T = 3;
};

enum NtpType{
    NTP_SERVER = 1;
    NTP_CLIENT = 2;
};

// 时间同步的以太网端口，当使用GNSS同步时，该端口应配置为与GNSS接收机的PPS信号所连接的芯片PTP_TIRG引脚一致
// 当PPS信号连接PTP_TRIG_IN00时，应配置为ETH_Port0
// 当PPS信号连接PTP_TRIG_IN10时，应配置为ETH_Port1
enum TimeSyncEthPort{

```



```

    ETH_Port0 = 1;
    ETH_Port1 = 2;
};

enum PtpType{
    PTP_SERVER = 1;
    PTP_CLIENT = 2;
};

// GNSS时间同步配置
message GnssTimeSyncConfig{
    // 配置是否使用GNSS方式时间同步, 默认为false
    optional bool use_gnss_timesync = 1 [default = false];
    // 配置接收机的通信端口, 不允许配置为default
    optional GnssReceiverPortType receiver_port = 2 [default = UART1];
    // 配置接收机的型号
    optional GnssReceiverModel receiver_model = 3 [default = BYNAV_BY682T];
}

// NTP客户端配置
message NtpClientConfig{
    // 配置提供NTP服务的远程服务器IP地址
    optional string ntp_server_ip = 1 [default = "192.168.2.6"];
    // 配置NTP客户端进行时间同步的周期, 单位为分钟
    optional uint32 update_period = 2 [default = 1];
}

// NTP服务器配置
message NtpServerConfig{
    // 配置ntp.conf文件所在目录, 默认该目录在BSTOS的/etc目录下
    optional string configfile_path = 1 [default = "/etc/ntp.conf"];
}

// NTP时间同步配置
message NtpTimeSyncConfig{
    // 配置是否使用NTP方式时间同步, 默认为false
    optional bool use_ntp_timesync = 1 [default = false];
    // 配置NTP的类型
    optional NtpType ntp_type = 2 [default = NTP_SERVER];
    // 当配置NTP类型为SERVER时, server_configuration中的配置生效
    optional NtpServerConfig server_configuration = 3;
    // 当配置NTP类型为CLIENT时, client_configuration中的配置生效
    optional NtpClientConfig client_configuration = 4;
}

// PTP的延迟计算方式选择
enum PtpDelayMechanism{
    PTP_E2E = 1; // PTP End-to-End方式
    PTP_P2P = 2; // PTP Peer-to-Peer方式, 需要PHY和SWITCH支持IEEE 1588V2
    PTP_AUTO = 3; // 自动, 从E2E方式开始, 当可以进行P2P方式时, 自动切换
};

// PTP网络传输方式选择
enum PtpNetworkTransport{
    IEEE_802_3 = 1; // MAC包
    UDP_IPV4 = 2; // IPV4
    UDP_IPV6 = 3; // IPV6
}

// PTP时间戳方式选择
enum PtpTimestampType{
    TS_HARDWARE = 1; // 使用硬件时间戳
    TS_SOFTWARE = 2; // 使用软件时间戳, 不推荐, 精度较低
}

// PTP时间同步配置
message PtpTimeSyncConfig{
    // 配置是否使用PTP方式时间同步, 默认为false
    optional bool use_ptp_timesync = 1 [default = false];
    // 配置PTP的类型, 默认为server

```

```

optional PtpType ptp_type = 2 [default = PTP_SERVER];
// 配置PTP的延迟计算方式
optional PtpDelayMechanism delay_mechanism = 3 [default = PTP_E2E];
// 配置PTP数据报文传输方式
optional PtpNetworkTransport network_transport = 4 [default = UDP_IPV4];
// 配置PTP的时间戳类型
optional PtpTimestampType timestamp_type = 5 [default = TS_HARDWARE];
}

message TimedConf {
    // GNSS时间同步配置
    optional GnssTimeSyncConfig gnss_timesync_configuration = 1;
    // NTP时间同步配置
    optional NtpTimeSyncConfig ntp_timesync_configuration = 2;
    // PTP时间同步配置
    optional PtpTimeSyncConfig ptp_timesync_configuration = 3;
    // 时间同步的网卡端口，必须配置
    required TimeSyncEthPort eth_port = 4 [default = ETH_Port0];
};

```

配置文件

示例的配置文件打包到系统镜像中，注意，在SDK中没有打包放置配置文件，用户需要修改配置时，可以根据需要直接在系统中修改。配置文件位于 `/etc/` 路径下。

- `timed_config.pb.txt`

```

``` gnss_timesync_configuration {

```

```

 use_gnss_timesync: false
 receiver_port: UART1
 receiver_model: BYNAV_BY682T

```

```

}

```

```

ntp_timesync_configuration {

```

```

 use_ntp_timesync: false
 ntp_type: NTP_SERVER
 server_configuration
 {
 configfile_path: "/etc/ntp.conf"
 }
 client_configuration
 {
 ntp_server_ip: "192.18.2.6"
 update_period: 1
 }
}

```

```

}

```

```

ptp_timesync_configuration {

```

```

 use_ptp_timesync: false
 ptp_type: PTP_SERVER
 delay_mechanism: PTP_E2E
 network_transport: UDP_IPV4
 timestamp_type: TS_HARDWARE

```

```

}

```

```

eth_port: ETH_Port0

```

```
- ntp.conf
```

**This is the most basic ntp configuration file**

**The driftfile must remain in a place specific to this machine - it records the machine specific clock error**

```
driftfile /var/lib/ntp/drift
```

**This should be a server that is close (in IP terms) to the machine. Add other servers as required.**

**Unless you un-comment the line below ntpd will sync**

**only against the local system clock.**

```
#
```

```
server time.server.example.com
```

```
#
```

**Using local hardware clock as fallback**

**Disable this when using ntpd -q -g -x as ntpdate or it will sync to itself**

```
server 127.127.1.0 fudge 127.127.1.0 stratum 14
```

**Defining a default security setting**

```
restrict default
```

使用adb或ssh远程连接到系统后，使用vi编辑器可以直接修改配置文件。

### 运行timed

要使用PTP同步时，请确保Server和Client之间的网络可用，且IP可以互相ping通。当A1000使用交换机时，不会自动获取IP地址，因此不

```
sh-4.4# systemctl stop NetworkManager sh-4.4# ifconfig eth0 0 # 根据需要选择eth0或eth1 sh-4.4# ifconfig eth0 192.168.2.3 # 根据需要选择eth0或eth1，根据需要配置IP地址 sh-4.4# ping 192.168.2.4 # Client ping server端的IP地址，能ping通说明配置成功
```

确保server端和client端可以ping通后，执行：

```
sh-4.4# chmod +x /usr/sbin/* sh-4.4# /usr/sbin/timed
```

timed启动后，可能会启动多个服务，因此要停止时间同步时，关闭timed并不能保证关闭了时间同步。更简单的方法是修改配置文件后，重

### 查看NTP同步状态

1，配置为NTP\_CLIENT时

当配置ntp\_type为NTP\_CLIENT时，启动timed后，会自动打印同步状态。此外，通过date命令，也可以查看当前的系统时间，对比A1000与server的

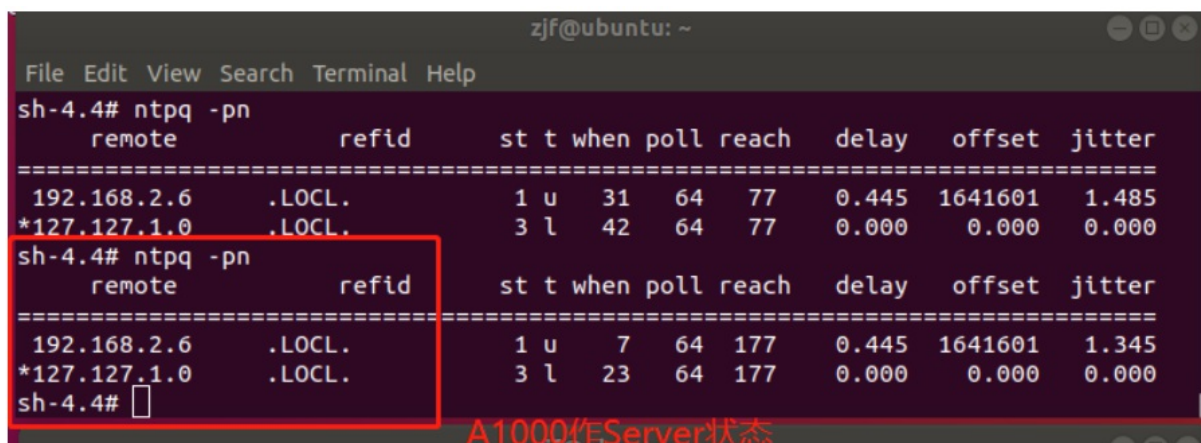
2，配置为NTP\_SERVER时

当配置ntp\_type为NTP\_SERVER时，NTP服务运行在后台。

使用ntpq工具可以查看NTP\_SERVER的状态：

```
sh-4.4# ntpq -pn ``
```

当ntpq输出的状态中，remote服务器前面有一个“\*”号，且reach的值达到177或以上时，表示时间同步服务稳定（在不稳定阶段也可以完成同步），如图所示。



```
zjf@ubuntu: ~
File Edit View Search Terminal Help
sh-4.4# ntpq -pn
 remote refid st t when poll reach delay offset jitter
=====
 192.168.2.6 .LOCL. 1 u 31 64 77 0.445 1641601 1.485
*127.127.1.0 .LOCL. 3 l 42 64 77 0.000 0.000 0.000
sh-4.4# ntpq -pn
 remote refid st t when poll reach delay offset jitter
=====
 192.168.2.6 .LOCL. 1 u 7 64 177 0.445 1641601 1.345
*127.127.1.0 .LOCL. 3 l 23 64 177 0.000 0.000 0.000
sh-4.4#
```

A1000作Server状态

- [sensor-manager-demo](#)

## sensor-manager-demo

目前有两个demo,分别为 `sensor_manager_demo` 和 `camDemo`

通过如下方法将demo的可执行文件以及诊断配置文件从SDK中拷贝到开发板系统/usr/bin目录中

```
cd {SDK_INSTALL_DIR}/sysroots/aarch64-bst-linux/usr/bin/
adb push sensor_manager_demo /usr/bin/
adb push sm_diag_com_config.pt /usr/bin/
```

`camdemo`按照同样方法操作

```
cd {SDK_INSTALL_DIR}/sysroots/aarch64-bst-linux/usr/bin/
adb push camDemo /usr/bin/
```

运行程序之前，加速度计和陀螺仪数据目前（2021-01-11）只能获取模拟数据，GNSS数据可以获取模拟或者实际物理设备数据。

注意，demo基于cyber库开发，运行demo之前需要执行如下命令

```
source /usr/bin/cyber/setup.bash
```

目前传感器的获取基于SOME-IP协议，也就是通过客户端-服务器模式获取数据，接收端为客户端，发送端为服务器端。因此，如果开机后Sensor服务没有自动启动的情况下，需要 运行demo前需要启动服务端。启动方法 参考如下命令

```
V SOMEIP_CONFIGURATION=/etc/bstos_service/sensorservice/vsomeip-sensor.json SensorService
```

注意，demo基于cyber库开发，运行demo之前需要执行如下命令

```
source /usr/bin/cyber/setup.bash
```

然后进入/usr/bin目录，demo启动命令如下：

```
V SOMEIP_CONFIGURATION=/etc/bstos_service/sensorservice/vsomeip-sensor.json ./sensor_manager_demo port
```

port 要根据硬件实际情况传入，目前 A1000板 传入2，FAD开发板传入5，后期会修正为从配置文件读入。

`camDemo` 不需要手动启动服务端，因为服务端开机启动运行，但USB相机目前不支持热插拔，所以启动系统之前要保证相机已经连到A1000板上，在/usr/bin目录下 直接 用如下命令启动即可

```
./camDemo
```

- 工具使用

## 工具使用

BST-Platform提供工具，协助开发者进行二次开发，调试和监控系统运行状态等，包括监控工具，录制工具，诊断转换工具等。

工具	用途	运行环境
Monitor工具	用于实时监控系统中指定的通信channel的相关信息	ARM Linux
SysMo工具	用于监控系统的任务调度情况	ARM Linux
Record工具	一个针对多传感器的数据录像回放工具	ARM Linux
诊断转换工具	用于转换诊断配置表的数据格式	X86 Ubuntu18.04

- [监控工具](#)
  - [介绍](#)
  - [使用说明](#)
  - [常用命令](#)
  - [UI界面快捷键](#)

## 监控工具

### 介绍

Monitor是一个基于ncurses库开发的一个文本用户界面工具，用于实时监控系统中指定的通信channel，可获取并显示channel的如下信息：

- channel的名称
- channel的帧率
- channel的消息类型
- channel的消息包大小
- channel的消息包的各个字段内容
- channel的readers和writers

### 使用说明

Monitor已集成进A1000镜像，所在路径：`/usr/bin/cyber/tools/cyber_monitor`，用户可通过命令行方式实现其支持的功能。

### 常用命令

命令	功能
<code>cyber_monitor -h</code>	查看帮助信息
<code>cyber_monitor -c channel_name</code>	监控channel_name通道的信息
<code>cyber_monitor</code>	未指定通道名字则监控所有通道

### UI界面快捷键

快捷键	功能
ESC / q	退出
Backspace	返回
H / h	查看帮助信息
F / f	第二列显示帧率
T / t	第二列显示通道消息类型
Space	关闭/打开channel，黄色为关闭，绿色为打开
PageDown / Ctrl + d	下一页
PageUp / Ctrl + u	上一页

Up / w	上一行
Down / s	下一行
右箭头 / Enter / d	选中当前channel，查看详细信息
左箭头 / a	返回上一层
l / i	(channel页面)显示channel的readers和writers
B / b	(channel页面)显示channel的消息内容



- [任务调度监控工具](#)
  - [介绍](#)
  - [使用](#)

## 任务调度监控工具

### 介绍

任务调度监控工具 **Sysmo tool**用于监控系统的任务调度情况，通过在任务调度的代码块里面插桩输出调度信息然后通过**UI**显示出来，便于分析查看。

为了不影响系统的执行性能，默认情况下插桩代码不会被执行。

任务的基本执行和调度单位是协程，协程可看作一种用户空间实现的轻量级线程，其运行载体仍然是**POSIX**线程。

**Sysmo Tool**提供两种类型的监控信息：

- 采样输出各个**POSIX**线程承载的各个任务的执行状态，采样间隔**100ms**
- 实时输出各个**POSIX**线程每一帧的执行内容与状态

### 使用

开启

- 采样输出： `export sysmo_start=1`
- 实时输出： `export sysmo_start=2`

关闭

- `export sysmo_start=0`

- 录制回放工具
  - 介绍
  - 使用说明
  - 常用命令

## 录制回放工具

### 介绍

**cyber\_recorder**是一个针对多传感器的数据录像回放工具。它提供了许多有用的功能，包括记录录像文件、重放录像文件、分割录像文件、查看录像文件的信息等。

### 使用说明

**cyber\_recorder**已集成进A1000镜像，所在路径：`/usr/bin/cyber/tools/cyber_recorder`，用户可通过命令行方式实现其支持的功能。

### 常用命令

- 查看录像文件的信息：

```
$ cyber_recorder info -h
usage: cyber_recorder info file
usage: cyber_recorder info [options]
-h, --help show help message # 显示帮助信息
```

- 录制录像文件

```
$ cyber_recorder record -h
usage: cyber_recorder record [options]
-o, --output <file> output record file # 输出录像文件
-a, --all all channels # 所有通道
-c, --channel <name> channel name # 通道名称
-i, --segment-interval <seconds> record segmented every n second(s) # 每隔n秒分段一次录像
-m, --segment-size <MB> record segmented every n megabyte(s) # 每隔n兆字节分段一次录像
-h, --help show help message # 显示帮助消息
```

- 播放录像文件

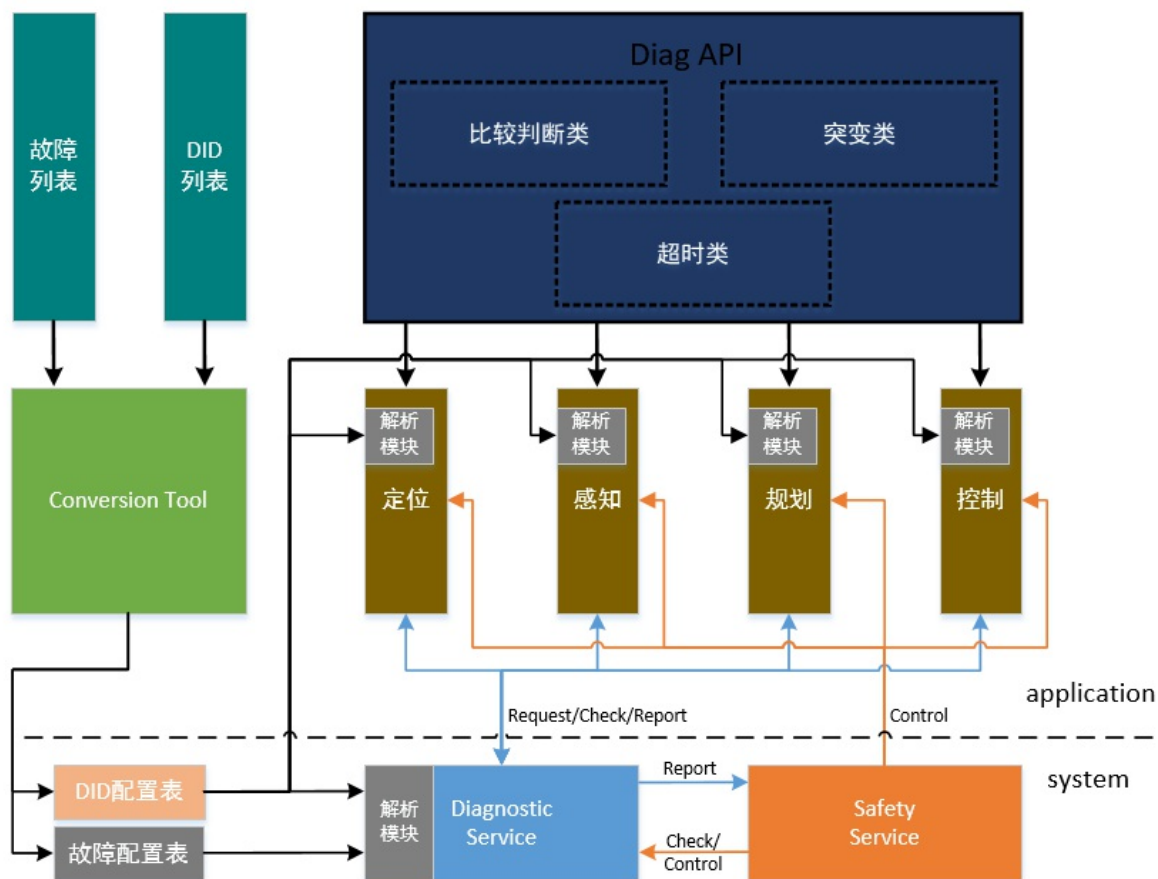
```
$ cyber_recorder play -h
usage: cyber_recorder play [options]
-f, --file <file> input record file # 输入录像文件
-a, --all play all # 播放所有
-c, --white-channel <name> only play the specified channel # 只播放指定的通道
-k, --black-channel <name> not play the specified channel # 不播放指定的通道
-l, --loop loop play # 循环播放
-r, --rate <1.0> multiply the play rate by FACTOR # 将播放速率乘以因数
-b, --begin <2018-07-01 00:00:00> play the record begin at # 从指定时刻播放录像
-e, --end <2018-07-01 00:01:00> play the record end at # 播放录像至指定时刻
-s, --start <seconds> play started at n seconds # n秒后开始播放
-d, --delay <seconds> play delayed n seconds # 播放延迟n秒
-p, --preload <seconds> play after trying to preload n second(s) # 尝试预加载n秒后播放
-h, --help show help message # 显示帮助消息
```

- 分割录像文件

```
$ cyber_recorder split -h
usage: cyber_recorder split [options]
-f, --file <file> input record file # 输入录像文件
-o, --output <file> output record file # 输出录像文件
-c, --white-channel <name> only split the specified channel # 只分割指定通道
-k, --black-channel <name> not split the specified channel # 不分割指定通道
在指定时刻开始（以字符串的形式，例如：“2018-07-01 00:00:00”）
-b, --begin <2018-07-01 00:00:00> begin at assigned time
在指定时刻结束（以字符串的形式，例如：“2018-07-01 00:00:00”）
-e, --end <2018-07-01 01:00:00> end at assigned time
-h, --help show help message # 显示帮助消息
```

- 故障诊断配置表及转换工具使用说明
  - 简介
  - 故障诊断配置表配置
  - 转换工具使用
  - 转换结果验证

## 故障诊断配置表及转换工具使用说明



### 简介

整个BST-Platform故障诊断服务可以分为3大组件，一部分为故障诊断配置表以及转换工具，负责故障诊断配置表生成以及将故障诊断配置表从excel表格到二进制文件的转换；一部分为二进制故障诊断配置表解析组件，负责解析提取二进制故障诊断配置表信息，并获取某一DTC或DID的信息；最后一部分为平台常用诊断算法API (diag API), 负责为应用提供基础常用的故障诊断算法。

本文档将描述如何使用故障诊断配置表以及转换工具实现故障诊断配置表生成以及将故障诊断配置表从excel表格到二进制文件的转换，主要包含两大内容：

- 故障诊断配置表配置
- 转换工具使用

### 故障诊断配置表配置

#### DTC列表内容

序号	故障名称	故障分类	模块ID(HEX)	故障码(HEX)	故障处理类型	故障等级	故障清除机制	故障记录条件	故障描述		备注
									故障成熟条件	故障解除条件	
ID	Name	Type	Module_ID	DTC	ProcessType	Level	Clear_Mechanism	Log_Condition	Fault_Confirmation	Troubleshooting	Remarks

为了方便用户阅读和修改故障列表，故障列表以excel表格形式设计，包含以下内容：

- 序号：从1开始增加，表征诊断故障码数量；
- 故障名称：诊断故障名；
- 故障分类：具有某种共同属性的故障归纳为一类故障，目前分为default\_error（默认错误）,communication\_error（通信错误）,hardware\_error（硬件错误）,function\_error（功能错误）,logic\_error（逻辑错误）五类；
- 模块ID：故障诊断所在模块的ID，分配规则后续提供；
- 故障码（DTC）：诊断故障码，分配规则后续提供；
- 故障处理类型：表征诊断故障发生时，BST-OS功能安全模块对其的处理措施，包括：INVALID（无效），IGNORE（忽略），RESTARTSERVICE（重启应用），RESTARTSYSTEM（重启系统）；
- 故障等级：划分为4级，从0开始，0级为OK，1级为warn，2级为error，3级为FATAL；
- 故障清除机制：将储存的故障码清除的机制，用户自定义；
- 故障记录条件：满足何种条件，故障码才会被记录，用户自定义；
- 故障成熟条件：即为故障产生原因，用户自定义；
- 故障解除条件：故障消除所需的必要条件，用户自定义；
- 备注：用以用户标注其他信息，用户自定义；

#### DID列表内容

序号	数据名称	模块ID号(HEX)	DID号(HEX)	DID数据长度(byte)	数据类型	读写权限	描述
ID	Name	Module_ID	DID	Size	Type	Access	Desc

DID（数据标识符）一般表示控制器相关信息：(1) 定义一些软硬件版本号、供应商信息、软件日期、车辆VIN码等静态数据信息；(2) 定义控制器运行状态信息，比如控制器温度、电压、电流等动态数据信息。与故障列表一样，DID列表同样以excel表格形式设计，包含以下内容：

- 序号：从1开始增加，表征DID数量；
- 数据名称：DID名称；
- 模块ID：DID所在模块的ID，分配规则后续提供；
- DID：全局数据标识符，分配规则后续提供；
- DID数据长度：DID所表征数据需要占据的字节长度；
- 数据类型：DID数据类型，包括：  
int8\_t,uint8\_t,int16\_t,uint16\_t,int32\_t,uint32\_t,int64\_t,uint64\_t,float,double,ASCII
- 读写权限：DID读写权限，包括：READ\_ONLY（只读）,WRITE\_ONLY（只写）,READ\_WRITE（可读写）；
- 描述：数据具体描述。

#### 使用说明

1. 打开conversion\_tool文件夹下diagnostic.xlsx表格；
2. 用户请遵照示例将自己模块的DTC诊断错误码信息填入diagnostic.xlsx表格fault\_list工作表中，将DID信息填

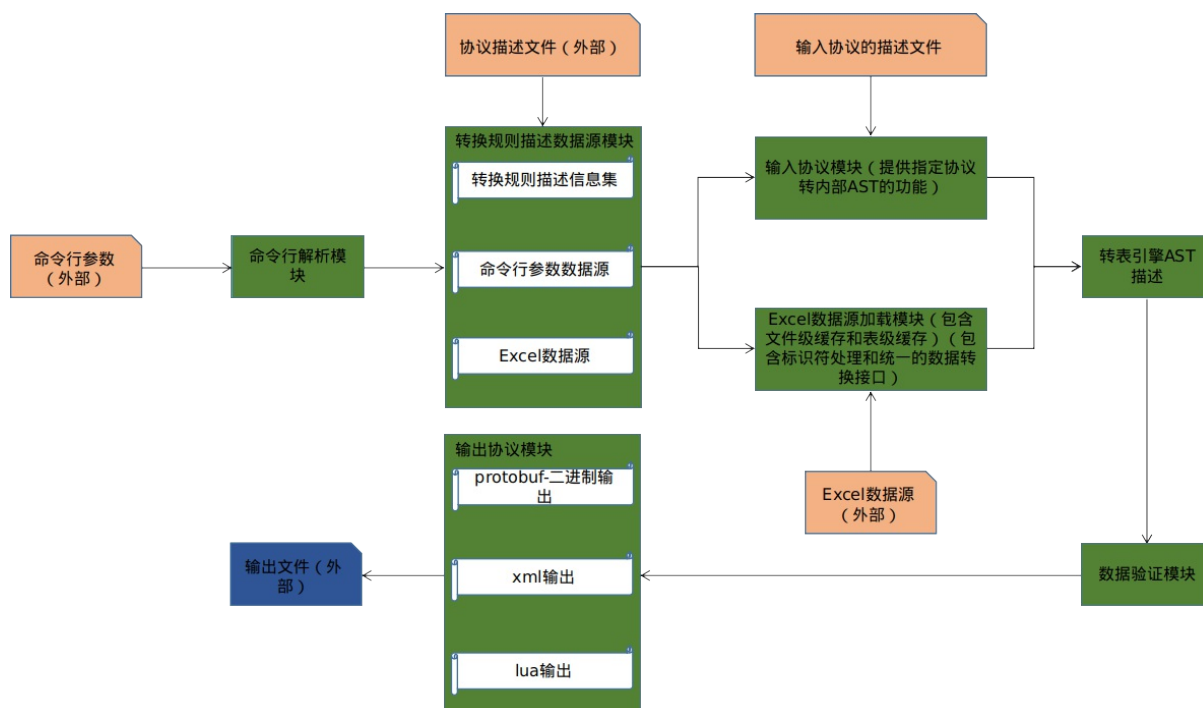
充入diagnostic.xlsx表格DID\_list工作表中;

3. 点击保存, 退出

#### 注意:

- 填充内容时只能使用英文填充, 不支持中文填充;
- 故障码、模块ID以及DID都以十六进制数字表示, 填充时请以“0x”开始填充, 故障码为三字节十六进制, 比如: 0x000001, DID为两字节十六进制, 比如: 0xFD00, 模块ID为一字节十六进制, 比如: 0x10;
- 转换工具不会把表格中所有元素进行转换, 当前fault\_list工作表中只转换ID、Name、Module\_ID、DTC、Level、Type、ProcessType七项元素, DID\_list工作表只转换ID、Name、Module\_ID、DID、Size、Type、Access七项元素;

## 转换工具使用



## 使用说明

1. 首先对conversion\_tool文件夹下dtc\_list\_output\_config.xml和did\_list\_output\_config.xml两个xml文件进行配置, 其中dtc\_list\_output\_config.xml为DTC列表转换配置, did\_list\_output\_config.xml为DID列表转换配置。更改以下配置项:

字段	简介	主配置	次配置	补充配置
work_dir	工作目录	默认为: 当前目录		
output_dir	输出目录	默认为: output文件夹		
DataSource	配置数据源	文件路径	表名 (DTC列表表名为 fault_list, DID列表表名为 DID_list)	数据起始行号, 列号 (英文逗号分隔)
OutputFile	输出文件	如: diagnostic_dtc_demo.bin		

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
 <globals>
 <work_dir desc="工作目录, 相对于当前xml的目录"></work_dir>
 <xresloader_path desc="xresloader地址, 相对于当前xml的目录">./xresconv-cli-1.3.1/xresloader-2.9.0.jar</xresloader_path>
 <proto desc="协议类型, -p选项">protobuf</proto>
 <output_type desc="输出类型, -t选项, 支持多个同时配置多种输出">bin</output_type>
 <!-- output_type desc="多种输出时可以额外定义某个节点的重命名规则" rename="/(?!)\.bin$/\.json/">json</output_type-->
 <!-- output_type desc="可以通过指定class来限制输出的规则" rename="/(?!)\.bin$/\.csv/" class="client" >ue-csv</output_type-->
 <!-- output_type 里的class标签对应下面item里的class标签, 均可配置多个, 多个用空格隔开, 任意一个class匹配都会启用这个输出 -->
 <proto_file desc="协议描述文件, -f选项">diagnostic.pb</proto_file>
 <output_dir desc="输出目录, -o选项">./output</output_dir>
 <data_src_dir desc="数据源目录, -d选项"></data_src_dir>
 <data_version desc="数据版本号, 留空则自动生成">1.0.0</data_version>
 <rename desc="重命名规则, 正则表达式: /搜索模式/替换内容/, 对应xresloader的-n选项, 如果在output_type里设置了rename, 以output_type里的rename为准, 否则使用这里的全局配置" placeholder="/(?!)\.bin$/\.json/"></rename>
 <java_option desc="java选项-最大内存限制2GB">-Xmx2048m</java_option>
 <java_option desc="java选项-客户端模式">-client</java_option>
 <default_scheme name="KeyRow" desc="默认scheme模式参数-Key行号">3</default_scheme>
 <!-- default_scheme name="MacroSource" desc="默认scheme模式参数-Key行号">资源转换示例.xlsx|macro|2,1</default_scheme-->
 </globals>
 <list>
 <item name="升级表" cat="kind" class="client server">
 <scheme name="DataSource" desc="数据源(文件名|表名|数据起始行号,数据起始列号)">diagnostic.xlsx|fault_list|4,1</scheme>
 <scheme name="ProtoName" desc="协议名">diagnostic_fault</scheme>
 <scheme name="OutputFile" desc="输出文件名">diagnostic_dtc.bin</scheme>
 </item>
 </list>
</root>
```

1. 在conversion\_tool文件夹下新建一个终端terminal, 执行以下命令:
2. DTC列表转换为二进制文件

```
python xresconv-cli-1.3.1/xresconv-cli.py dtc_list_output_config.xml
```

3. DID列表转换为二进制文件

```
python xresconv-cli-1.3.1/xresconv-cli.py did_list_output_config.xml
```

4. 若列表转换没有出错, 终端上将打印出以下内容:

```
python ./xresconv-cli-1.3.1/xresconv-cli.py did_list_output_config.xml
[NOTICE] start to run conv cmds on dir: /home/bst-qiu/workspace/test_xreloader/diagnostic_conversion/conversion_tool
[TRACE] xresloader - convert from "diagnostic.xlsx|DID_list" to "diagnostic_did.bin" started (protocol=diagnostic_did) ...
[INFO] xresloader - convert from "diagnostic.xlsx|DID_list" to "diagnostic_did.bin" success.(charset: utf-8)
[INFO] all jobs done. 0 job(s) failed.
```

若列表转换出错, 终端上将打印出以下内容, 具体错误信息记录在xresloader.run.log。

```
python ./xresconv-cli-1.3.1/xresconv-cli.py fault_list_output_config.xml
[NOTICE] start to run conv cmds on dir: /home/bst-qiu/workspace/test_xreloader/diagnostic_conversion/conversion_tool
[INFO] all jobs done. 1 job(s) failed.
```

1. 转换成功后, 二进制文件输出保存在output文件夹下。

备注

该转换工具采用开源转表工具xresloader, 需进一步了解转换工具使用, 请查看xresloader官方文档: [https://xresloader.atframe.work/zh\\_CN/latest/index.html](https://xresloader.atframe.work/zh_CN/latest/index.html)

## 转换结果验证

验证程序放置于/test下, 在/test下新建一个终端terminal, 运行以下命令:

```
//验证DTC列表转换是否成功
./DTC_Convert_Check [dtc.bin]
//验证DID列表转换是否成功
./DID_Convert_Check [did.bin]
```

eg:

```
./DTC_Convert_Check ../output/diagnostic_dtc_demo.bin
./DID_Convert_Check ../output/diagnostic_did_demo.bin
```